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
ДЗИСЮК Владислав
ТЕЛУХ Анастасія

КИЇВ – 2024

Варіант 8

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.

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 $pq \leq p_1q_1$; 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 М і знайти криптограму для абонентів А и 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

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.***

   Для виконання цього завдання ми вирішили обрати пошук випадкового числа із заданого інтервалу – створили функцію find_prime. Спочатку користувач має ввести початковий і кінцевий інтервал, потім за допомогою random.randint з цього інтервалу обирається випадкове число.

   Наступним кроком нам треба перевірити чи обране число просте. Для цього воно проходить дві перевірки – тест пробних ділень і тест Міллера-Рабіна.

   Першим у хід йде тест пробних ділень. За його допомоги, ми можемо попередньо легко перевірити чи обране число просте – шляхом спроби поділу на низку малих простих множників:

```
def trial_div(N):
    small_prime = [2, 3, 5, 7, 11]
    for prime_num in small_prime:
        if pascal_div(N, prime_num):
            return False
    return True
```

   Варто згадати, що для побудови тесту пробних ділень ми використали ознаку подільності Паскаля – функція pascal_div. Саме ця функція дозволяє перевірити подільність числа N на m, не виконуючи прямого ділення Nmodm:

```python
def pascal_div(N, m, B=10):
    num_part = []
    while N > 0:
        num_part.append(N % B)
        N //= B

    r = [1]
    for i in range(1, len(num_part)):
        r.append((r[i - 1] * B) % m)

    excess = sum(d * r[i] for i, d in enumerate(num_part)) % m
    return excess == 0
```

Якщо обране число пройшло тест пробних ділень, то далі слідує наступна перевірка – тест Міллера-Рабіна. Ми намагалися реалізувати цей тест згідно з усією вказаною теоретично інформацією у методичних вказівка, отримавги отаку функцію:

```python
#Тест Міллера-Рабіна
1 usage
def mil_rab_test(p, k=10):
    if p < 2 or p % 2 == 0:
        return p == 2
    s = 0
    d = p - 1
    while d % 2 == 0:
        d //= 2
        s += 1

    def check(x):
        x = pow(x, d, p)
        if x == 1 or x == p - 1:
            return False
        for _ in range(s - 1):
            x = pow(x, 2, p)
            if x == p - 1:
                return False
        return True
```

```python
counter = 0
for _ in range(k):
    x = random.randint( a: 2, p - 2)
    if gcd(x, p) != 1:
        return False
    if check(x):
        return False
    counter += 1
if counter < k:
    return False
return True
```

Отримуємо такий результат:

```
Початок діапазону: 50
Кінець діапазону: 2070
Генерація простого числа...
Просте число: 449
```

Бачимо, що 449 – просте число, яке і справді лежить у заданому діапазоні.

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 A, p_1 і q_1 – абонента B.***

У нашому коді це реалізовує функція get_two. Вона генерує два простих числа довжиною 256 біт для кожного абонента: p, q – для абонента A і p_1, q_1 – для абонента B:

```
Пари простих чисел p і q:
Абонент A: p = 856512108580244853270920108833959540997932009828800285588883704748654596140 99, q =
 713468052031786426009465238778997668125920204147788883651067128525336823138 51
Абонент B: p1 = 100573957610868710884814890650648183969160598792431375904282612807353113439039, q1 =
 958721325292547395242444430216380569552971075635015203134971884657243784072 73
```

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 d_1.***

   Згідно з методичними вказівками, ми знаємо, що для отримання ключів нам треба мати прості числа p і q, модуль n (n = p×q), функцію Ойлера φ(n) (φ(n) = (p-1)(q-1)).

   Оскільки ми вже створили функцію генерації простих чисел p і q, а модуль n можна безпосередньо обрахувати у функції генерації ключів, нам необхідно було створити функцію Ойлера:

```python
def func_oil(p, q):
    return (p - 1) * (q - 1)
```

   Також ми знаємо, що для обрахунку секретного ключа d нам необхідно знайти обернене до e за mod φ(n). Оскільки функцію Ойлера ми вже можемо обрахувати, то можемо і створити функцію для знаходження обернених чисел за модулем:

```python
def inverse(a, m):
    gcd, x, _ = adv_eucl(a, m)
    if gcd != 1:
        return None
    return x % m
```

   Взявши стандартне значення для e = 2^{16}+1 (що рівне числу 65537), ми змогли реалізувати функцію генерацію ключів RSA:

```python
def keys_rsa(bit_leng=256):
    (p, q), _ = get_two(bit_leng=256)
    n = p * q
    phi = func_oil(p, q)
    e = 65537
    if gcd(e, phi) != 1:
        raise ValueError("Значення e має бути взаємнопростим із функцією Ойлера!")
    d = inverse(e, phi)
    return (d, p, q), (n, e)
```

Спробуємо:

```
Абонент A: Секретний ключ: d =
 28787258882622554452901125348997562271212556433525060688434815067956339313817212920549140087648196534168283753831946437729200690973233508441846664274051 73,
 Відкритий ключ: n =
 611094025650385239911826208012584860094081336761549542427996137436807116124020067104436747049488251739909845885661113069303509513728743382511894786258524 9, e =
 65537
Абонент B: Секретний ключ: d1 =
 774312889007735561275053973511199319083402238190521279665384346292386924876150398791662310146686854572445161355906915774934966146544206389517442385243518 5,
 Відкритий ключ: n1 =
 964223979306085342288153152482537570061542732795458646476852949950866668483332431962168600501445481131724380380387863489800177004799107098100376239973064 7, e1 =
 65537
```

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.***

Операція шифрування виконується за формулою $C = M^e \bmod n$:

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

Операція розшифрування виконується за формулою $M = C^d \bmod n$:

```python
def decrypt(cipher, secret_key):
    d, p, q = secret_key
    n = p * q
    return pow(cipher, d, n)
```

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

```python
def digit_signature(M, secret_key):
    d, p, q = secret_key
    n = p * q
    return pow(M, d, n)
```

Перевірка цифрового відбувається за формулою: $M = S^e \bmod n$:

```python
def ver_signature(M, digit_signature, public_key):
    n, e = public_key
    return pow(digit_signature, e, n) == M
```

Повідомлення від абонентів генеруються як числа в діапазоні від 1000 до 9999 включно:

```python
a_M = random.randint( a: 1000,  b: 9999)
b_M = random.randint( a: 1000,  b: 9999)
```

Поглянемо на результат:

```
Повідомлення від A до B: 9182
Зашифроване:
 712636607942879131949404607032978801166479123558121082129283567498759590331925309170953004399403078020494449610664678865527523606638648989279351685177 69
Розшифроване:
 360115628367700767060485037111779573726617573357563663819457696088989304236793611471291083991084759768368816772159497515113014881643588385798359459568769 6
-----
Повідомлення від B до A: 5148
Зашифроване:
 611027795447291911386953750537664332322676648152225501547565111706962864968675911465776193084201955190653095288708032139643235748885113707902346179886665 0
Розшифроване:
 224829988746393813320249396269867547831737569356802810452838067617475612437670910925899797344552562171603737096445017159578332839620022372660343101939938 7
```

Перевірка підписів:

```
Цифровий підпис від A:
 173499239377571105582185983992671954551814693481671216801524246630593082171299657387244792775727290639890101016061081894030509777794147748535052552384378 6
Цифровий підпис від B:
 816758926901654615446247402724589529401503591817059112011118383421078189040440609365678064088960087737433686026954140765737664598395298152719026239714378 9
Перевірка підпису A: Коректний
Перевірка підпису B: Коректний
```

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.***

Для цього ми створили дві функції: k_send для відправки ключа і k_receive для його отримання.

Для відправки відбуваються такі процеси: кшифрується відкритим ключем отримувача -> створюється цифровий підпис ключа за допомогою секретного ключа відправника -> підпис шифрується відкритим ключем отримувача.

Для отримання: отриманий зашифрований ключ розшифровується за допомогою секретного ключа отримувача -> зашифрований цифровий підпис розшифровується за допомогою секретного ключа отримувача -> перевіряється правильність цифрового підпису.

```python
def k_send(k, send_secr_k, send_publ_k, rec_publ_k):
    key_encr = encrypt(k, rec_publ_k)
    key_signature = digit_signature(k, send_secr_k)
    encr_signature = encrypt(key_signature, rec_publ_k)
    return key_encr, encr_signature


#Отримання ключа
1 usage
def k_receive(encr_secr_k, encr_signature, rec_secr_k, send_publ_k):
    key_decr = decrypt(encr_secr_k, rec_secr_k)
    decr_signature = decrypt(encr_signature, rec_secr_k)
    check_signature = ver_signature(key_decr, decr_signature, send_publ_k)
    return key_decr, check_signature
```

Перевіримо результат:

```
Переданий ключ від A до B: 23663
Зашифрований ключ:
 884115280488037872712416183994098171234973181868435301917085394170654863289748875296649753296222980824158702994213925359440215359623581734997720487983146
Зашифрований підпис:
 834130448459919225798209493607359788983962169518374606670823809112464396750694209139161792395588651898129060841575883224139221691696745271188457657775279
Розшифрований ключ: 23663
Підтвердження підпису: Вірний
```

Наостанок, перевіримо роботу наших функцій шляхом використання стороннього ресурсу - https://www.dcode.fr/rsa-cipher :

Бачимо, що результати виявилися однаковими, тому код працює правильно.

Висновок. Отже, ми ознайомилися із принципами роботи криптосистеми RSA і навіть спробували реалізувати її власноруч. При виконанні цього практикуму найскладнішим виявилася реалізація тестувань для виявлення простого числа. Спочатку ми не використовували тестування пробних ділень, що призводило до того, що код працював довше і важче. Проте, звісно, проблема була вирішена швидко завдяки уважному аналізу методичних вказівок.