



Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Фізико-технічний інститут

ЛАБОРАТОРНА РОБОТА №5

з дисципліни

«Криптографія»

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

Виконали:

студенти 3 курсу ФТІ

групи ФБ-73

Маковецький Андрій та Бадарак Оксана

Перевірили:

Чорний О.

Савчук М. М.

Завадська Л. О.

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 $p \cdot q \leq p_1 \cdot q_1$; 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 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 $0 < k < n$.

Кожна з наведених операцій повинна бути реалізована у вигляді окремої процедури, інтерфейс якої повинен приймати лише ті дані, які необхідні для її роботи; наприклад, функція Encrypt(), яка шифрує повідомлення для абонента, повинна приймати на вхід повідомлення та відкритий ключ адресата (і тільки його), повертаючи в якості результату шифротекст. Відповідно, програмний код повинен містити сім високорівневих процедур: GenerateKeyPair(), Encrypt(), Decrypt(), Sign(), Verify(), SendKey(), ReceiveKey().

Кожну операцію рекомендується перевіряти шляхом взаємодії із тестовим середовищем, розташованим за адресою <http://asymcryptwebservice.appspot.com/?section=rsa>. Наприклад, для перевірки коректності операції шифрування необхідно а) зашифрувати власною реалізацією повідомлення для серверу та розшифрувати його на сервері, б) зашифрувати на сервері повідомлення для вашої реалізації та розшифрувати його локально.

Результати

Message:

k = 1337

User 1 private key:

d1:

33700669191226032601970721994340458180474246954607607147468756939587655263534729381
23670468544813955823816514911741178038701376499047523050393313405752373

User 1 public key:

n1:

55057727951773164617608256446294169457154199742343730515359838552904304200581750036
39032677922459110863685035768634028234731223303400497988001342106914557

e1:

65537

User 2 private key:

d2:

15537003701004754451700838429871286713114327524740626252163176116936866953305602066
57059719675078126623662986988208589649266674889556226231586166980485473

User 2 public key:

n2:

10126788777252596643472082030616355219466670144096732199781383124571769761499846527
986585526323381346979905773178745397936677896357209776095523866872405383

e2:

65537

Signature:

40557716546226145215897388054527350986062794608963994157500459537178880631806240072
99516172795056534020034165906354267070095013557109895254779550598606612

Код програми

```
import random
from math import gcd

def modular_inverse(a, b):
    x, y = 0, 1
    u, v = 1, 0
    m = b
    a1 = a
    b1 = b

    while a != 0:
        q = b // a
        r = b % a

        m = x - u * q
        n = y - v * q

        b, a, x, y, u, v = a, r, u, v, m, n

    gcd = b
```

```

    if x < 0:
        x += m

    if gcd == 1:
        return x
    else:
        raise ValueError('Modular inverse for such values does not exist:', a1, b1)
        #print('Modular inverse for such values does not exist:', a1, 'mod', b1)
        #return False

def prime_by_miller_rabin(p, rounds=1024):

    # simple prime division tests
    primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 27, 29, 31, 37]

    if p in primes or p == 1:
        return True

    for pr in primes:
        if p % pr == 0:
            return False

    d = p - 1
    s = 0

    while d % 2 == 0:
        d //= 2
        s += 1

    for i in range(rounds):

        x = random.randrange(2, p)

        if gcd(x, p) > 1:
            return False

        x = pow(x, d, p)

        if x in (1, p - 1):
            continue

        xr = x

        for r in range(1, s):
            xr = pow(xr, 2, p)
            if xr == p - 1:
                return True
            if xr == 1:
                return False

        if xr != p - 1:
            return False

    return True

def create_random_prime(min, max):

    x = random.randrange(min, max + 1)
    m0 = 0
    if x % 2 == 0:
        m0 = x + 1
    else:
        m0 = x

    #print(x, m0)

    for i in range((max - m0) // 2):
        p = m0 + 2 * i
        #print(p)
        if prime_by_miller_rabin(p):
            return p

```

```

        else:
            return create_random_prime(min, max)

def create_prime(bit_count):
    return create_random_prime(2 ** (bit_count - 1), (2 ** bit_count) - 1)

def generate_RSA_key_pair(p, q):

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

    e = create_random_prime(2, eul - 1)
    e = (2 ** 16) + 1

    while gcd(e, eul) != 1:
        e = create_random_prime(2, eul - 1)

    try:
        d = modular_inverse(e, eul) % eul
    except ValueError as ve:
        print(ve)

    return d, n, e

def RSA_encrypt(message, e, n):
    return pow(message, e, n)

def RSA_decrypt(ciphertext, d, n):
    return pow(ciphertext, d, n)

def RSA_sign(message, d, n):
    return message, pow(message, d, n)

def RSA_validate_signature(msg, sign, e, n):
    return True if msg == pow(sign, e, n) else False

# TODO: Add checking for n1 >= n
def RSA_send_key(e, n, d, e1, n1, k):

    k1 = pow(k, e1, n1)
    s = pow(k, d, n)
    s1 = pow(s, e1, n1)

    return k1, s1

def RSA_receive_key(k1, d1, n1, s1, e, n):

    k = pow(k1, d1, n1)
    s = pow(s1, d1, n1)

    if RSA_validate_signature(k, s, e, n) == False:
        raise Exception('RSA key not validated!')

    return k

#print(prime_by_miller_rabin(7571, 128))
#print(create_random_prime(1, 10000))
#print(create_prime(256))

def main():

    d1, n1, e1 = generate_RSA_key_pair(create_prime(256), create_prime(256))
    d2, n2, e2 = generate_RSA_key_pair(create_prime(256), create_prime(256))
    msg = 1337

```

```

print('User 1 private key:\n', d1)
print('User 1 public key:')
print('\n1\n', n1)
print('e1\n', e1)

print('\nUser 2 private key:\n', d2)
print('User 2 public key:')
print('\n2\n', n2)
print('e2\n', e2)

sign = RSA_sign(msg, d1, n1)[1]
print('Signature:', sign)
print('\nSent:', msg)

encr = RSA_encrypt(msg, e1, n1)
k1, s1 = RSA_send_key(e1, n1, d1, e2, n2, msg)
k = RSA_receive_key(k1, d2, n2, s1, e1, n1)
print(RSA_validate_signature(k, sign, e1, n1))

print('Received:', k)

main()

```

Висновки:

Під час данного комп'ютерного практикуму, ми ознайомились з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA. Також, використовуючи криптосистему типу RSA, організували канал засекреченого зв'язку й електронний підпис, ознайомились із протоколом розсилання ключів.