**Лабораторна робота №3**

Скоробагатько Максим, ФБ-31мн

**Тема**: Реалізація основних асиметричних криптосистем.

**Мета**: Дослідження можливостей побудови загальних та спеціальних криптографічних протоколів за допомогою асиметричних криптосистем.

RSA (Rivest–Shamir–Adleman) — це криптографічний алгоритм з відкритим ключем (асиметричний алгоритм), що забезпечує шифрування та цифровий підпис. Основна ідея RSA базується на складності факторизації великих чисел, що робить його надійним для захисту даних.

В основі RSA лежить розділення ключів:

- Приватний ключ (Private Key) — зберігається в секреті, доступний лише власнику.
- Публічний ключ (Public Key) — відкрито розповсюджується для всіх бажаючих.

### Основний принцип:

- Шифрування виконується за допомогою публічного ключа, але розшифрування можливе тільки за допомогою приватного ключа.
- Цифровий підпис генерується приватним ключем, але перевіряється за допомогою публічного ключа.

Таким чином, ключі є взаємозалежними, але зворотне обчислення приватного ключа з публічного є обчислювально складним завданням.

### Крок 1: Генерація ключів

1. Вибір двох простих чисел $p$ і $q$ великого розміру.
2. Обчислення модуля $n$: $n = p \times q$. Значення $n$ використовується як частина обох ключів.
3. Обчислення функції Ейлера: $\varphi(n) = (n - 1) \times (q - 1)$.

4. Вибір відкритого експонента $e$ має бути взаємно простим із $\varphi(n)$.
5. Обчислення закритого експонента $d$:$d = e^{-1} mod \varphi(n)$.

Публічний ключ: $(e, n)$.
Приватний ключ: $(d, n)$.

### Шифрування даних

1. Повідомлення $M$ перетворюється у числову форму.
2. Шифрування виконується за формулою: $C = M^e mod n$, де: $C$ — шифротекст, ,$e$ і $n$ — частини публічного ключа.

Таким чином, шифрування можна виконати, знаючи тільки публічний ключ.

### Розшифрування даних

1. Отриманий шифротекст $C$ розшифровується за допомогою приватного ключа: $M = C^d mod n$ де: $d$ і $n$ — частини приватного ключа.

Лише той, хто володіє приватним ключем, може виконати розшифрування.

### Цифровий підпис в RSA

RSA також дозволяє створювати цифрові підписи для забезпечення автентичності та цілісності даних:

1. Створення підпису:
    - Повідомлення хешується за допомогою геш-функції (наприклад, SHA-256).
    - Хеш шифрується приватним ключем для створення підпису.
2. Перевірка підпису:
    - Підпис розшифровується за допомогою публічного ключа.
    - Розшифрований хеш порівнюється з хешем оригінального повідомлення.

Це дозволяє перевірити, що:

- Дані не було змінено.
- Підпис належить власнику приватного ключа.

Реалізація на Python

```
import math as m
import numpy as np
import random

def find_s(n):
    count = 0
    temp = n - 1
    flag = 0
    if n%2 == 0:
        return False,0
    else:
        while flag == 0:
            temp /= 2
            count+=1
            if temp%2 == 1:
                break
    return count,int((n-1)/(2**count))

def miller_rabin_test(n: int):
    x = 0
    k = 400
    s,t = find_s(n)
    if not(s):
        return True
    for i in range(k):
        a = random.randrange(2, n-2)
        x = pow(a, t, n)
        if x == 1 or x == n-1:
            continue
        for i in range(s-1):
            x = (x**2) % n
            if x == 1:
                return False
```

```
            if x == n-1:
                break
        if x==n-1:
            continue
        return False
    return True

def gen_pr():
    n_0 = 2**32+1
    n_1 = int((2**64 - 2**32)/2)
    for i in range(0,n_1):
        x = random.randrange(n_0, n_1)
        if x%2 == 0:
            x+=1
        x+=2*i
        if miller_rabin_test(x):
            break
    return x

def gen_keys():
    p,q,p_1,q_1 = 1,1,1,1
    while p*q >= p_1 * q_1:
        p_1 = gen_pr()
        q_1 = gen_pr()
        p = gen_pr()
        q = gen_pr()
    return p, q, p_1, q_1

def fi_n(p, q):
    return (p - 1) * (q - 1)

def key_generation(p, q):
    n = p * q
    e = pow(2, 16) + 1
    d = pow(e, -1, fi_n(p, q))
    return [(d, p, q), (n, e)]

def encryption(M, ne):
    n, e = ne[0], ne[1]
    return pow(M, e, n)
```

```
def decryption(C, dpq):
    d = dpq[0]
    n = dpq[1] * dpq[2]
    return pow(C, d, n)

def digital_signification(M, dpq):
    d, n = dpq[0], dpq[1] * dpq[2]
    return (M, pow(M, d, n))

def verify(MS, ne):
    M, S = MS[0], MS[1]
    n, e = ne[0], ne[1]
    return M == pow(S, e, n)

def send_message(M):
    p,q,p_1,q_1 = gen_keys()
    keys_A = key_generation(p,q)
    C = encryption(M, keys_A[1])
    print("Send message: ", hex(C)[2:], "\nKeys:", [hex(i)[2:] for i in
keys_A[0]] )
    return C, keys_A[0]

def read_message(C, keys):
    Message = decryption(C, keys)
    return Message

class mes():
    def __init__(self, name,p,q):
        self.name = name
        self.p = p
        self.q = q
    def gen_data(self):
         self.k = random.randrange(0, 2**32)
         self.k_1 = pow(self.k,self.e_1,self.n_1)
         self.S = pow(self.k,self.keys[0][0],self.keys[1][0])
         self.S_1 = pow(self.S,self.e_1,self.n_1)
    def verify(self, k, S):
        k_ver = pow(S, self.keys[1][1],self.keys[1][0])
        return k_ver == k
```

```
    def get_data(self, k_1, S_1):
        k = pow(k_1, self.keys_first[0][0], self.keys_first[1][0])
        S = pow(S_1, self.keys_first[0][0], self.keys_first[1][0])
        return k,S
    def get_open_key(self, e_1, n_1):
        self.e_1 = e_1
        self.n_1 = n_1
    def gen_keys(self):
        self.keys = key_generation(self.p,self.q)
        while self.keys[1][0] >= self.n_1:
            print(self.keys[1][0], self.n_1)
            self.keys = key_generation(self.p,self.q)
    def gen_first_keys(self):
        self.keys_first = key_generation(self.p,self.q)
def main():
    M = random.randrange(0, 2**32)
    print("Generate_message: ", hex(M)[2:])
    cr_mess, k = send_message(M)
    M_decr = decryption(cr_mess, k)
    print("Decryption message:", hex(M_decr)[2:])

    print("============================")
    M = random.randrange(0, 2**32)
    print("Generate_message: ", hex(M)[2:])
    p,q = gen_keys()[0:2]
    keys = key_generation(p,q)
    dig_sign = digital_signification(M,keys[0])
    print("Digital signification:",
hex(dig_sign[0])[2:],hex(dig_sign[1])[2:])
    print("Verification:",verify(dig_sign,keys[1]))
    print("============================")
    print("Send keys")
    p,q,p_1,q_1 = gen_keys()
    A = mes("A",p,q)
    B = mes("B",p_1,q_1)
    B.gen_first_keys()
    A.get_open_key(B.keys_first[1][1], B.keys_first[1][0])
    A.gen_keys()
    A.gen_data()
    k,S = B.get_data(A.k_1,A.S_1)
```

```
    print(A.verify(k,S))

def print_result(M,p,q):
        keys = key_generation(p, q)
        print("n = ", hex(keys[1][0])[2:])
        print("e = ", hex(keys[1][1])[2:])
        print("M = ", hex(M)[2:])
        C = int("0x"+input("Ciphertext = "), 16)
        res = decryption(C, keys[0])
        print("Decrypt = ",hex(res)[2:])

        n = int("0x"+input("Modulus = "), 16)
        e = int("0x"+input("E = "), 16)
        encryp = encryption(M,[n, e])
        print("Encrypt = ", hex(encryp)[2:])
        sign = int("0x"+input("Sign = "), 16)
        print(verify([M,sign],[n, e]))
        sign_my = digital_signification(M, keys[0])
        print("My sign: ", hex(sign_my[1])[2:])

if __name__ == "__main__":
    main()
```

Результати роботи:

```
thon311/python.exe "g:/навчання/кпі/3 сем/методи реалізації криптографічних механізмів/3/lab3.py"
Generate_message:  cdb1d35a
Send message:  1597ff70489b03d1a6f82f6d920ff
Keys: ['578aad41dc1f3f4f345dc2699001', 'f2b6f55ad8dbc01', '18c2e1d6106c97']
Decryption message: cdb1d35a
==========================
Generate_message:  a0fe351c
Digital signification: a0fe351c 416f3c8f856e7a409135dab13bd
Verification: True
==========================
Send keys
True
PS G:\навчання\кпі\3 сем\методи реалізації криптографічних механізмів>
```

**Висновок:**

**RSA** є одним із найпопулярніших асиметричних криптографічних алгоритмів, який використовується для **шифрування даних** та **цифрових**

**підписів**. У цьому звіті розглянемо основні аспекти використання RSA на практиці, включно з генерацією ключів, шифруванням, розшифруванням та особливостями його застосування.