Лабораторна робота з криптографії №4

Виконав: Костюковець Остап ФБ-96

Варіант №5

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

Мета: 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

Завдання 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.

In [1]:

```
import random


def gcd(a, b):
    p = [0, 1]
    gcd_val = b
    a, b = max(a, b), min(a, b)

    while b != 0:
        q = a // b
        gcd_val = b
        a, b = b, a % b
        p.append(p[-1] * (-q) + p[-2])

    return gcd_val, p[-2]


def miller_rabin(n, k):
    if n == 2:
        return True

    if n % 2 == 0:
        return False

    r, s = 0, n - 1
    while s % 2 == 0:
        r += 1
```

```
        s //= 2
    for _ in range(k):
        a = random.randrange(2, n - 1)
        x = pow(a, s, 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 random_prime(n):
    min_number = 2 ** (n - 1)
    max_number = 2 ** (n + 1) - 1
    random_number = random.randint(min_number, max_number)

    if not random_number & 1:
        random_number += 1

    while True:
        if miller_rabin(random_number,40):
            return random_number
        else:
            random_number += 2


print(hex(random_prime(255)))
```

```
0xed31b20ff04364519c978a698535c3ac2e558931fedf24b9d4e6f5fe7dcb2b2f
```

Завдання 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 < p 1 q 1 ; p і q – прості числа для побудови ключів абонента A, p 1 і q 1 – абонента B.

In [2]:

```
size = 255
p, q = random_prime(size), random_prime(size)
p1, q1 = random_prime(size), random_prime(size)

if p*q>p1*q1:
    temp = p
    p = p1
    p1 = temp

    temp = q
    q = q1
    q1 = temp

print(f" p = {hex(p)}\n q = {hex(q)} \n p1 = {hex(p1)}\n q1 = {hex(q1)}")
```

```
p = 0x83d6eb04b9b8525e2a6a7f9e3fb0afc7518910e3cb070d87c1cabf957e3ee5e9
q = 0x6a7376e087ad4bad0b4476f67140bdc8c1d12723e77db68fa6db1210dfd70993
p1 = 0x7d7c9511cfec389829c01ae5b66081368b06d0ab32f7f94aa202615fe97f04af
q1 = 0x9a39d0485639c278fdd2064331ce15a8db84aab160b9bd3808cf5d86e8c18d39
```

Завдання 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 .

In [3]:
```
def RSA(p, q):
    e = 0x10001

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

    _, d = gcd(e, f)

    return d % f, p, q, n, e

d, _, _, n, e = RSA(p,q)
d1, _, _, n1, e1 = RSA(p1,q1)


print(f" d = {hex(d)}\n n = {hex(n)} \n e = {hex(e)}\n")
print(f" d1 = {hex(d1)}\n n1 = {hex(n1)} \n e1 = {hex(e1)}")
```

```
 d = 0x31a09c223f8b6f8b8ca2d0561e11ce344f07f2774b8918a2b06b0825880fa49c86eb965f713d6729a
73e682e0d423e15bc9f0949423c17efe30a30642329a5b1
 n = 0x36d274143a4726cada2308ae7e93ee29791ee81aacfe2414e81f9f8d52dbd23213d1b67ac78d3d738
5030c66584de25413dfa1b8649941319c19d5f0c1e235cb
 e = 0x10001

 d1 = 0xf9fe19da3cdad4b0d0e9c7f54515f956426b513c9b86375cfd4de3e1a2dd83b1f45dfdb59ccf7580
d2f83d275afbe77875d5661525b07e58a09658c68848a91
 n1 = 0x4b9948829677f5bb700381c8f0dfd96bb4b454309685cbd702f13d93554b6653fa7e3a488bf315fb
1bbb33051037494649ed82dbc4c8b80d55c4ad9112cb6df7
 e1 = 0x10001
```

Завдання 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.

In [4]:
```
def encrypt(m, e, n):
    return pow(m, e, n)


def decrypt(c, n, d):
    return pow(c, d, n)


def sign(m, n, d):
    return pow(m, d, n)


def verify(m, s, e, n):
    return m == pow(s, e, n)
```

Завдання 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 .

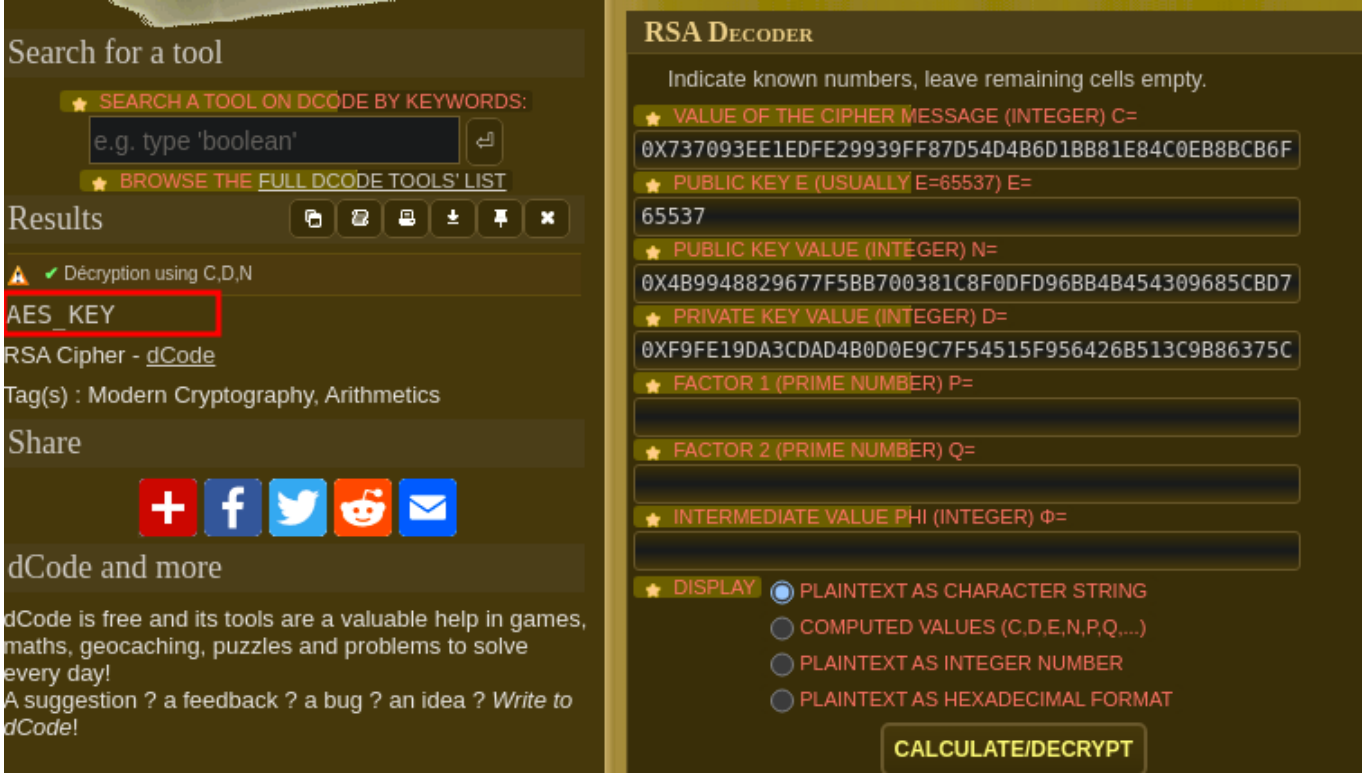
In [5]:
```
def send_key(k, A_private, B_public):
    c = encrypt(k, *B_public)
    s = sign(c, *A_private)
    s1 = encrypt(s, *B_public)
    return c, s1


def recv_key(c, s1, A_public, B_private):
    s = decrypt(s1, *B_private)
    if verify(c, s, *A_public):
        k = decrypt(c, *B_private)
        return k
    return None
```

Перевірка локально

In [6]:
```
m = int(b"AES_KEY".hex(), 16)

# A has: p, q, d, e, n, e1, n1, m
# B has: p1, q1, d1, e1, n1, e, n, c, s1

c, s1 = send_key(m, [n, d], [e1, n1])
print(f"Encrypted : {hex(c)}")

m = recv_key(c, s1, [e, n], [n1, d1])
print(f"Decrypted : {bytes.fromhex(hex(m)[2:])}")
```

```
Encrypted : 0x737093ee1edfe29939ff87d54d4b6d1bb81e84c0eb8bcb6fc3674b65b1f29c26b7262d17e0
b88f90135c15df2f9c8e29360e3615e37b84f30d32a0bbe2f29e1
Decrypted : b'AES_KEY'
```

Перевірка онлайн



Висновок

> Під час виконання даної лабораторної роботи я ознайомився з тестами перевірки чисел на простоту і методами генерації ключів для фсиметричної криптосистеми типу RSA. Також, я ознайомися з криптосистемою RSA та реалізував засекречений зв'язок з використанням цієї системи