readme

倪浚桐

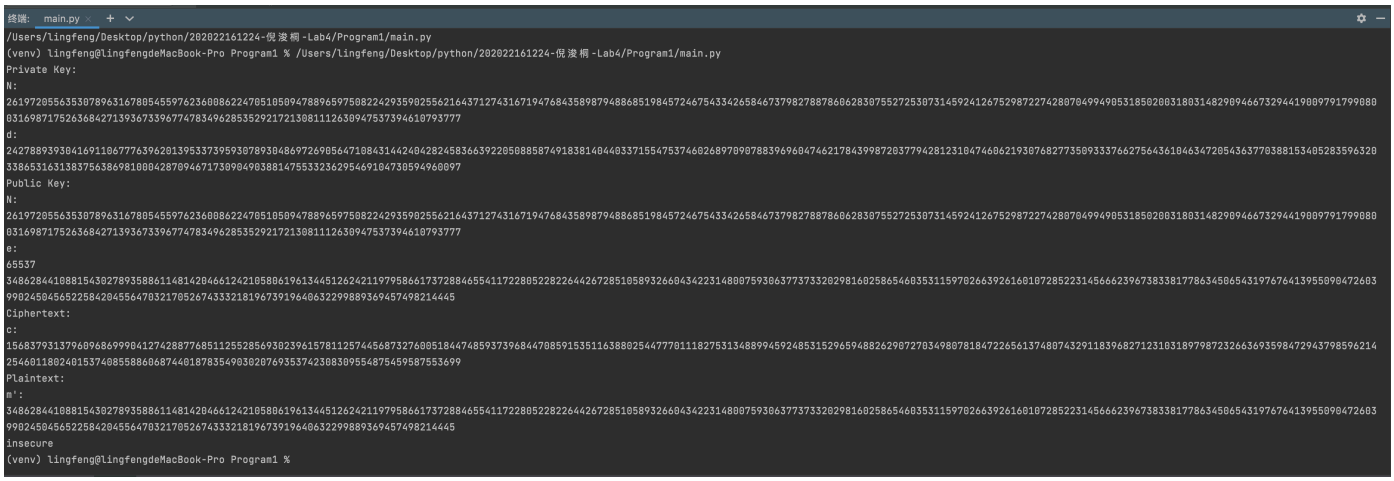
202022161224

Lab4-progarm1

macOS Monterey 12.0.1

Pycharm 11.0.12 x86-64

Python 3.9.5




```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import random
import Prime


def encryption(message: int, puk: list) -> int:
    return Prime.quick_pow_mod(message, puk[1], puk[0])


def decryption(secret: int, prk: list) -> int:
    return Prime.quick_pow_mod(secret, prk[1], prk[0])


def get_RSAKey():
    RSAKey = {}
    prime_arr: list = Prime.get_rand_prime_arr(2)
    p: int = prime_arr[0]
    q: int = prime_arr[1]
    while p == q:
```

```
        q = random.choice(prime_arr)
    n: int = p * q
    s: int = (p - 1) * (q - 1)
    e: int = 65537
    d: int = Prime.mod_inverse(e, s)
    print("Private Key:")
    print("N:")
    print(n)
    print("d:")
    print(d)
    print("Public Key:")
    print("N:")
    print(n)
    print("e:")
    print(e)
    puk: list = [n, e]
    prk: list = [n, d]
    RSAKey['puk'] = puk
    RSAKey['prk'] = prk
    return RSAKey


if __name__ == '__main__':
    RSAKey: [str, list] = get_RSAKey()
    message: int = int(input())
    secret: int = encryption(message, RSAKey['puk'])
    print("Ciphertext:")
    print("c:")
    print(secret)
    message: int = decryption(secret, RSAKey['prk'])
    print("Plaintext:")
    print("m':")
    print(message)
    print("insecure")

```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import math
import random


# 扩展欧几里得算法求模反元素
def ex_euclid(a: int, b: int, list):
    if b == 0:
        list[0] = 1
        list[1] = 0
        list[2] = a
    else:
```

```
        ex_euclid(b, a % b, list)
        temp = list[0]
        list[0] = list[1]
        list[1] = temp - a // b * list[1]


# 求模反元素
def mod_inverse(a: int, b: int) -> int:
    list = [0, 0, 0]
    if a < b:
        a, b = b, a
    ex_euclid(a, b, list)
    if list[1] < 0:
        list[1] = a + list[1]
    return list[1]


# 快速幂模运算，把b拆分为二进制，遍历b的二进制，当二进制位为0时不计入计算
def quick_pow_mod(a: int, b: int, c: int) -> int:
    a = a % c
    ans: int = 1
    while b != 0:
        if b & 1:
            ans = (ans * a) % c
        b >>= 1
        a = (a % c) * (a % c)
    return ans


# n为要检验的大数，a < n,k = n - 1
def miller_rabin_witness(a: int, n: int) -> bool:
    if n == 1:
        return False
    if n == 2:
        return True
    k: int = n - 1
    q: int = int(math.floor(math.log(k, 2)))
    m: int = 1
    while q > 0:
        m = k // 2 ** q
        if k % 2 ** q == 0 and m % 2 == 1:
            break
        q = q - 1
    if quick_pow_mod(a, n - 1, n) != 1:
        return False
    b1: int = quick_pow_mod(a, m, n)
    for i in range(1, q + 1):
        if b1 == n - 1 or b1 == 1:
            return True
```

```
        b2: int = b1 ** 2 % n
        b1 = b2
    if b1 == 1:
        return True
    return False


# Miller-Rabin素性检验算法,检验8次
def prime_test_miller_rabin(p: int, k: int) -> bool:
    while k > 0:
        a: int = random.randint(1, p - 1)
        if not miller_rabin_witness(a, p):
            return False
        k = k - 1
    return True


# 判断 num 是否与 prime_arr 中的每一个数都互质
def prime_each(num: int, prime_arr: list) -> bool:
    for prime in prime_arr:
        remainder: int = num % prime
        if remainder == 0:
            return False
    return True


# return a prime array from begin to end
def get_con_prime_array(begin: int, end: int) -> list:
    array: list = []
    for i in range(begin, end):
        flag: bool = judge_prime(i)
        if flag:
            array.append(i)
    return array


# judge whether a number is prime
def judge_prime(number: int) -> bool:
    temp: int = int(math.sqrt(number))
    for i in range(2, temp + 1):
        if number % i == 0:
            return False
    return True


# 根据 count 的值生成若干个与质数数组都互质的大数
def get_rand_prime_arr(count: int) -> list:
    arr: list = get_con_prime_array(2, 100000)
    prime: list = []
```

```
    while len(prime) < count:
        num: int = random.randint(pow(10, 154), pow(10, 155))
        if num % 2 == 0:
            num = num + 1
        while True:
            if prime_each(num, arr) and prime_test_miller_rabin(num, 8):
                if num not in prime:
                    prime.append(num)
                break
            num = num + 2
    return prime

```