readme

倪浚桐

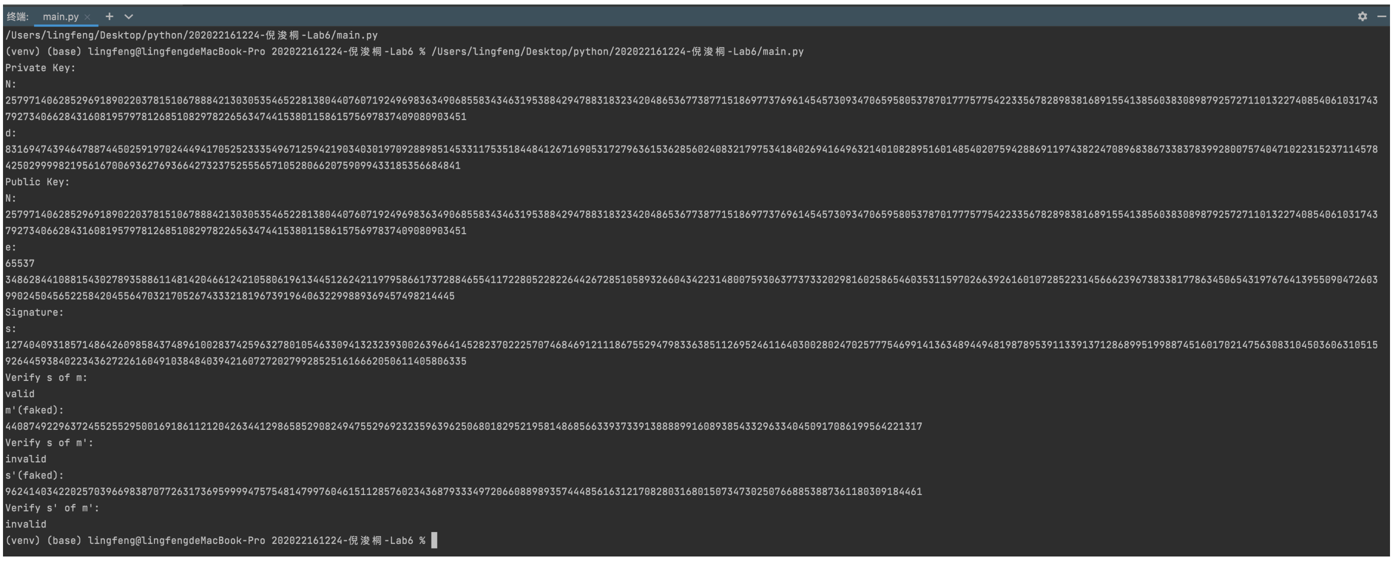
202022161224

Lab6-progarm1

macOS Monterey 12.0.1

Pycharm 11.0.12 x86-64

Python 3.9.5



```
/Users/lingfeng/Desktop/python/202022161224-倪浚桐-Lab6/main.py
(venv) (base) lingfeng@lingfengdeMacBook-Pro 202022161224-倪浚桐-Lab6 % /Users/lingfeng/Desktop/python/202022161224-倪浚桐-Lab6/main.py
Private Key:
N:
2579714062852969189022037815106788842130305354652281380440760719249698363490685583434631953884294788318323420486536773877151869773769614545730934706595805378701777577542233567828983816891554138560383089879257271101322740854061031743
792734066284316081957978126851082978226563474415380115861575697837409080903451
d:
8316947439464788744502591970244494170525233354967125942190340301970928898514533117535184484126716905317279636153628560240832179753418402694164963214010828951601485402075942886911974382247089683867338378399280075740471022315237114578
42502999982195616700693627693664273237525556571052806620759099433185356684841
Public Key:
N:
2579714062852969189022037815106788842130305354652281380440760719249698363490685583434631953884294788318323420486536773877151869773769614545730934706595805378701777577542233567828983816891554138560383089879257271101322740854061031743
792734066284316081957978126851082978226563474415380115861575697837409080903451
e:
65537
3486284410881543027893588611481420466124210580619613445126242119795866173728846554117228052282264426728510589326604342231480075930637737332029816025865460353115970266392616010728522314566623967383381778634506543197676413955090472603
990245045652258420455647032170526743332181967391964063229988936945749821444 5
Signature:
s:
1274040931857148642609858437489610028374259632780105463309413232393002639664145282370222570746846912111867552947983363851126952461164030028024702577754699141363489449481987895391133913712868995199887451601702147563083104503606310515
926445938402234362722616049103848403942160727202799285251616662050611405806335
Verify s of m:
valid
m'(faked):
44087492296372455255295001691861121204263441298658529082494755296923235963962506801829521958148685663393733913888899160893854332963340450917086199564221317
Verify s of m':
invalid
s'(faked):
96241403422025703966983870772631736959999475754814799760461511285760234368793334972066088989357444856163121708280316801507347302507668853887361180309184461
Verify s' of m':
invalid
(venv) (base) lingfeng@lingfengdeMacBook-Pro 202022161224-倪浚桐-Lab6 % 
```

```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__':
    # Generate a textbook RSA key pair. Print the private key and the public key as multiple decimal strings.
    RSAKey: [str, list] = get_RSAKey()

    # Read a decimal string representing a plaintext message . Raise an exception if is invalid.
    message: int = int(input())

    # Sign the message . Print the signature as a decimal string.
    secret: int = encryption(message, RSAKey['prk'])
    print("Signature:")
    print("s:")
    print(secret)

    # Verify the signature of message . Print valid if the signature is valid. Print invalid otherwise.
    message1: int = decryption(secret, RSAKey['puk'])
    print("Verify s of m:")
    if message1 == message:
        print("valid")
    else:
        print("invalid")

    # Randomly pick a number as a faked message , and verify the signature s of message m'
    # Print valid if the signature is valid. Print invalid otherwise.
    print("m'(faked):")
    m_fake = Prime.get_rand_prime_arr(2)
    print(m_fake[1])
    secret_fake: int = encryption(m_fake[1], RSAKey['prk'])
    print("Verify s of m':")
    if secret == secret_fake:
        print("valid")
    else:
        print("invalid")

    # Randomly pick a number as a faked signature , and verify the signature s' of message m'
    # Print valid if the signature is valid. Print invalid otherwise.
    print("s'(faked):")
    s_fake = Prime.get_rand_prime_arr(2)
    print(s_fake[1])
    message_fake: int = decryption(s_fake[1], RSAKey['prk'])
    print("Verify s' of m':")
    if message == message_fake:
        print("valid")
    else:
        print("invalid")

```

```python
#!/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

```