

readme

倪浚桐

202022161224

Lab4-progarm1

macOS Monterey 12.0.1

Pycharm 11.0.12 x86-64

Python 3.9.5

```
终端 main.py + -
/Users/Lingfeng/Desktop/python/202022161224-倪浚桐-Lab4/Program1/main.py
(venv) Lingfeng@LingfengdeMacBook-Pro Program1 % /Users/Lingfeng/Desktop/python/202022161224-倪浚桐-Lab4/Program1/main.py
Private Key:
N:
261972855635307896316780545597623608622470510509478896597508224293590255621643712743167194768435898794886851984572467543342658467379827887860628307552725307314592412675298722742807049949053185020031803148290946673294419089791799080
03169871752636842713936733967747834962853529217213081112630947537394610793777
d:
24278893938416911867776396201395337395930789304867726905647188431442404282458363922850885874918381404403371554753746026897090788396904746217843998720377942812310474606219307682773509333766275643610463472854363778388153485283596320
33865316313837563869810084287094671730904903881475533236295469104730594960097
Public Key:
N:
261972855635307896316780545597623608622470510509478896597508224293590255621643712743167194768435898794886851984572467543342658467379827887860628307552725307314592412675298722742807049949053185020031803148290946673294419089791799080
03169871752636842713936733967747834962853529217213081112630947537394610793777
e:
65537
3486284410881543027893588611481420466124210508619613445126242119795866173728846554117228052282264426728510589326604342231480075930637737332029816025865460353115970266392616010728522314566623967383381778634506543197674413955098472603
9902450456522584204556470321705267433321819673919640632299889369457498214445
Ciphertext:
c:
1568379313796096869998412742887685112552856930239615781125744568732760951844748593739694470859153511638802544777011182753348894592485315296594882629072703498078184722656137480743291183968271231031897987232663693598472943798596214
25460118024015374085588606874401878354903020769353742308389554875459587553699
Plaintext:
m":
3486284410881543027893588611481420466124210508619613445126242119795866173728846554117228052282264426728510589326604342231480075930637737332029816025865460353115970266392616010728522314566623967383381778634506543197674413955098472603
9902450456522584204556470321705267433321819673919640632299889369457498214445
insecure
(venv) Lingfeng@LingfengdeMacBook-Pro Program1 %
```

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  import random
4  import Prime
5
6
7  def encryption(message: int, puk: list) -> int:
8      return Prime.quick_pow_mod(message, puk[1], puk[0])
9
10
11  def decryption(secret: int, prk: list) -> int:
12      return Prime.quick_pow_mod(secret, prk[1], prk[0])
13
14
15  def get_RSAKey():
16      RSAKey = {}
17      prime_arr: list = Prime.get_rand_prime_arr(2)
18      p: int = prime_arr[0]
19      q: int = prime_arr[1]
20      while p == q:
```

```

21     q = random.choice(prime_arr)
22     n: int = p * q
23     s: int = (p - 1) * (q - 1)
24     e: int = 65537
25     d: int = Prime.mod_inverse(e, s)
26     print("Private Key:")
27     print("N:")
28     print(n)
29     print("d:")
30     print(d)
31     print("Public Key:")
32     print("N:")
33     print(n)
34     print("e:")
35     print(e)
36     puk: list = [n, e]
37     prk: list = [n, d]
38     RSAKey['puk'] = puk
39     RSAKey['prk'] = prk
40     return RSAKey
41
42
43 if __name__ == '__main__':
44     RSAKey: [str, list] = get_RSAKey()
45     message: int = int(input())
46     secret: int = encryption(message, RSAKey['puk'])
47     print("Ciphertext:")
48     print("c:")
49     print(secret)
50     message: int = decryption(secret, RSAKey['prk'])
51     print("Plaintext:")
52     print("m':")
53     print(message)
54     print("insecure")
55

```

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  import math
4  import random
5
6
7  # 扩展欧几里得算法求模反元素
8  def ex_euclid(a: int, b: int, list):
9      if b == 0:
10         list[0] = 1
11         list[1] = 0
12         list[2] = a
13     else:

```

```

14     ex_euclid(b, a % b, list)
15     temp = list[0]
16     list[0] = list[1]
17     list[1] = temp - a // b * list[1]
18
19
20 # 求模反元素
21 def mod_inverse(a: int, b: int) -> int:
22     list = [0, 0, 0]
23     if a < b:
24         a, b = b, a
25     ex_euclid(a, b, list)
26     if list[1] < 0:
27         list[1] = a + list[1]
28     return list[1]
29
30
31 # 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
32 def quick_pow_mod(a: int, b: int, c: int) -> int:
33     a = a % c
34     ans: int = 1
35     while b != 0:
36         if b & 1:
37             ans = (ans * a) % c
38             b >>= 1
39         a = (a % c) * (a % c)
40     return ans
41
42
43 # n为要检验的大数, a < n, k = n - 1
44 def miller_rabin_witness(a: int, n: int) -> bool:
45     if n == 1:
46         return False
47     if n == 2:
48         return True
49     k: int = n - 1
50     q: int = int(math.floor(math.log(k, 2)))
51     m: int = 1
52     while q > 0:
53         m = k // 2 ** q
54         if k % 2 ** q == 0 and m % 2 == 1:
55             break
56         q = q - 1
57     if quick_pow_mod(a, n - 1, n) != 1:
58         return False
59     b1: int = quick_pow_mod(a, m, n)
60     for i in range(1, q + 1):
61         if b1 == n - 1 or b1 == 1:
62             return True

```

```

63         b2: int = b1 ** 2 % n
64         b1 = b2
65         if b1 == 1:
66             return True
67         return False
68
69
70 # Miller-Rabin素性检验算法, 检验8次
71 def prime_test_miller_rabin(p: int, k: int) -> bool:
72     while k > 0:
73         a: int = random.randint(1, p - 1)
74         if not miller_rabin_witness(a, p):
75             return False
76         k = k - 1
77     return True
78
79
80 # 判断 num 是否与 prime_arr 中的每一个数都互质
81 def prime_each(num: int, prime_arr: list) -> bool:
82     for prime in prime_arr:
83         remainder: int = num % prime
84         if remainder == 0:
85             return False
86     return True
87
88
89 # return a prime array from begin to end
90 def get_con_prime_array(begin: int, end: int) -> list:
91     array: list = []
92     for i in range(begin, end):
93         flag: bool = judge_prime(i)
94         if flag:
95             array.append(i)
96     return array
97
98
99 # judge whether a number is prime
100 def judge_prime(number: int) -> bool:
101     temp: int = int(math.sqrt(number))
102     for i in range(2, temp + 1):
103         if number % i == 0:
104             return False
105     return True
106
107
108 # 根据 count 的值生成若干个与质数数组都互质的大数
109 def get_rand_prime_arr(count: int) -> list:
110     arr: list = get_con_prime_array(2, 100000)
111     prime: list = []

```

```
112     while len(prime) < count:
113         num: int = random.randint(pow(10, 154), pow(10, 155))
114         if num % 2 == 0:
115             num = num + 1
116         while True:
117             if prime_each(num, arr) and prime_test_miller_rabin(num, 8):
118                 if num not in prime:
119                     prime.append(num)
120                 break
121             num = num + 2
122     return prime
123
```