

readme

倪浚桐

202022161224

Lab4-progarm2

macOS Monterey 12.0.1

Pycharm 11.0.12 x86-64

Python 3.9.5

```
终端: main.py x + v
/Users/lingfeng/Desktop/python/202022161224-倪浚桐-Lab4/Program2/main.py
(venv) lingfeng@lingfengdeMacBook-Pro Program2 % /Users/lingfeng/Desktop/python/202022161224-倪浚桐-Lab4/Program2/main.py
4137696876930890267522398697653550193405311689664069574322834683213199126531348263326633721504049779673544721298253021191958429503842792929508773630980912
Private Key:
p:
11798427372728096379290021397510692875108125360021207815012128324895864985223023955168700768399390870402518292203525382010422897360409791591753156557493681
alpha:
2694168472545231258806031947729903595445496088726119862765572815664866464580698723996864932479737768585376978355220734629128446893069598555763781086823030
a:
10216475235792255573243169432021589161939552450463032680788641049536011084645495238176466703573703908250052171007247192302314651903578109090596806665327191
Public Key:
p:
11798427372728096379290021397510692875108125360021207815012128324895864985223023955168700768399390870402518292203525382010422897360409791591753156557493681
alpha:
2694168472545231258806031947729903595445496088726119862765572815664866464580698723996864932479737768585376978355220734629128446893069598555763781086823030
beta:
7427531840129648840633167627025647456924364716746163773241706978207962017254660258880406761312616326530349230196214259678639839888312355048194890062570759
Ciphertext:
r:
9688458763014259035787617288156398771557104001789160054543902402508353287358720425831358264346126091462780281393232547180039504962280806782144535538996686
t:
5900614277258128539048043992956720894460134081839743517885672251548880941684093872871535484806495001285093686099836592822927411999498483545744456603867612
Plaintext:
4137696876930890267522398697653550193405311689664069574322834683213199126531348263326633721504049779673544721298253021191958429503842792929508773630980912
(venv) lingfeng@lingfengdeMacBook-Pro Program2 %
```

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  import random
4  import Prime
5
6
7  def gcd(a: int, b: int) -> int:
8      if a < b:
9          return gcd(b, a)
10     elif a % b == 0:
11         return b
12     else:
13         return gcd(b, a % b)
14
15
```

```

16 def gen_key(p: int) -> int:
17     key: int = random.randint(pow(10, 154), p)
18     while gcd(p, key) != 1:
19         key = random.randint(pow(10, 154), p)
20     return key
21
22
23 def power(a: int, b: int, c: int) -> int:
24     x: int = 1
25     y: int = a
26     while b > 0:
27         if b % 2 != 0:
28             x = (x * y) % c
29             y = (y * y) % c
30             b = int(b / 2)
31     return x % c
32
33
34 def encrypt(msg: str, p: int, beta: int, alpha: int) -> [int, list]:
35     en_msg: list = []
36
37     i: int = gen_key(p)
38     Km: int = power(beta, i, p)
39     Ke: int = power(alpha, i, p)
40
41     for i in range(0, len(msg)):
42         en_msg.append(msg[i])
43
44     for i in range(0, len(en_msg)):
45         en_msg[i] = Km * ord(en_msg[i])
46
47     return Ke, en_msg
48
49
50 def decrypt(t: list, r: int, a: int, p: int) -> list:
51     dr_msg = []
52     Km = power(r, a, p)
53     for i in range(0, len(t)):
54         dr_msg.append(chr(int(t[i] / Km)))
55
56     return dr_msg
57
58
59 def main():
60     msg: str = input()
61     prime_arr: list = Prime.get_rand_prime_arr(1)
62     p: int = prime_arr[0]
63     alpha: int = random.randint(2, p)
64     a: int = gen_key(p)

```

```

65     beta: int = power(alpha, a, p)
66
67     print("Private Key:")
68     print("p:")
69     print(p)
70     print("alpha:")
71     print(alpha)
72     print("a:")
73     print(a)
74     print("Public Key:")
75     print("p:")
76     print(p)
77     print("alpha:")
78     print(alpha)
79     print("beta:")
80     print(beta)
81
82     r, t = encrypt(msg, p, beta, alpha)
83
84     print("Ciphertext:")
85     print("r:")
86     print(r)
87     print("t:")
88     print(t[0])
89
90     dr_msg: list = decrypt(t, r, a, p)
91     d_msg: str = ''.join(dr_msg)
92
93     print("Plaintext:")
94     print(d_msg)
95
96
97 if __name__ == '__main__':
98     main()
99

```

```

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
```