|  |
| --- |
| 哈尔滨工业大学（深圳） |
| **《密码学基础》实验报告** |
|  |
| RSA密码算法实验  学 院: 计算机科学与技术   |  |  | | --- | --- | | 姓 名: | 李子韬 | | 学 号: | 220110609 | | 专 业: | 计算机科学与技术 | | 日 期: | 2024-10-16 | |

# 实验步骤

**请说明你的字符分组方式，以及关键的算法例如扩展欧几里德，素数检测，快速幂等函数的实现。**

**字符分组方式：两个字符的ascii码拼接为一个6位的十进制数字作为一个分组**

* **具体来说，对于加密，先从明文取两个字符，利用ord取ascii码，再通过ord0 \* 1000 + ord1实现拼接：**

m\_group = ord(pair[0]) \* 1000 + ord(pair[1])

**对于孤立的单个字符，使用\0补全：**

m\_group = ord(*plaintext*[i]) \* 1000

* **对于解密，也有类似操作，不过使用chr将ascii转为字符**

decrypted\_pair = mod\_pow(num, d, n)

dechar\_1 = chr(decrypted\_pair // 1000)

dechar\_2 = chr(decrypted\_pair % 1000)

**关键算法：**

* **欧几里得算法求gcd: 根据辗转相除法的定义，最终b==0时a就是最后的gcd，也就是最后一步前的b必能整除a(最极端情况下b=1)**

def gcd(*a*, *b*):

*while* *b*:

*a*, *b* = *b*, *a* % *b*

*return* *a*

* **素数检测：使用Miller-Rabin算法检测素数，思路就是先做初始检查（小边界和提高效率），之后分解n-1=2^r \* d的形式，进行k=10轮平方测试，每一轮测试中，依次检验平方模是否为n – 1，如果没有(for循环正常结束)，则调用for-else语句的else返回合数的判断，否则算暂时通过，进行下一轮的测试；**

def is\_prime(*n*, *k*=10):  *# 使用Miller-Rabin算法检测素数*

*if* *n* <= 1:

*return* False

*if* *n* <= 3:

*return* True

*if* *n* % 2 == 0:

*return* False

*# 写成 n - 1 = d \* 2^r*

    r, d = 0, *n* - 1

*while* d % 2 == 0:

        d //= 2

        r += 1

*# k次测试*

*for* \_ *in* range(*k*):

        a = random.randint(2, *n* - 2)

        x = pow(a, d, *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

* **扩展欧几里德算法求逆：思n路是每一轮的辗转相除需要得到q整除结果，然后利用x(n)=x(n-2)-qx(n-1)迭代即可，不过需要注意初始设置x的值即可（此处假定a是更大的那个数，故x0 = 0， x1 = 1，其中x0比x1更晚更新）**

def mod\_inv(*a*, *m*):

    m0, x0, x1 = *m*, 0, 1

*if* *m* == 1:

*return* 0

*while* *a* > 1:

        q = *a* // *m*

*m*, *a* = *a* % *m*, *m*

        x0, x1 = x1 - q \* x0, x0

*if* x1 < 0:

        x1 += m0

*return* x1

* **快速幂函数：从末位开始解析，每解析一个bit就让b平方模一次，同时让exp右移1位，实现快速幂计算**

def mod\_pow(*b*, *exp*, *mod*):

    res = 1

*b* = *b* % *mod*  *# b < mod*

*while* *exp* > 0:

*if* (*exp* % 2) == 1:

            res = (res \* *b*) % *mod*

*exp* = *exp* // 2

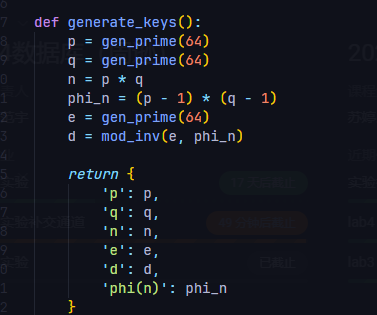
*b* = (*b* \* *b*) % *mod*

*return* res

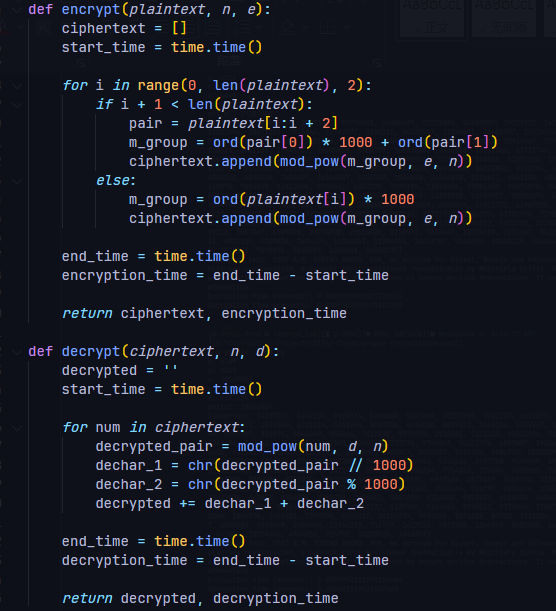
算法总体框架：

1. 生成p、q，计算n=pq、phi(n)=(p-1) \* (q-1)
2. 生成公钥e，mod\_inv计算模phi(n)逆元私钥d
3. 对每个明文分组，加密
4. 解密，解分组
5. 输出结果

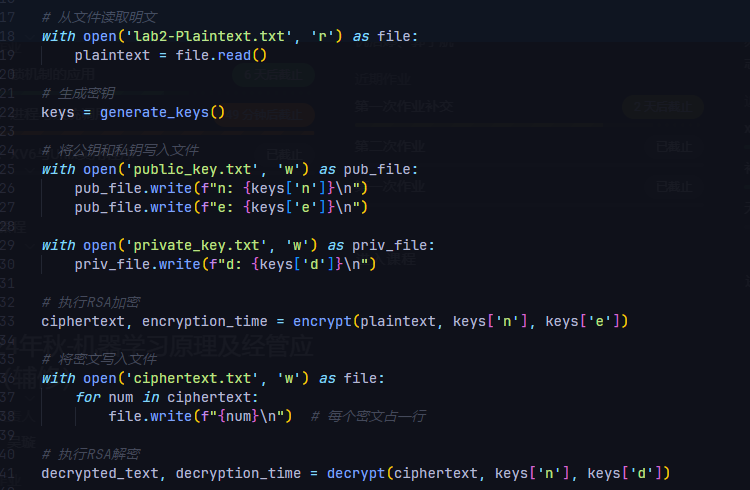
以下是实现：生成密钥

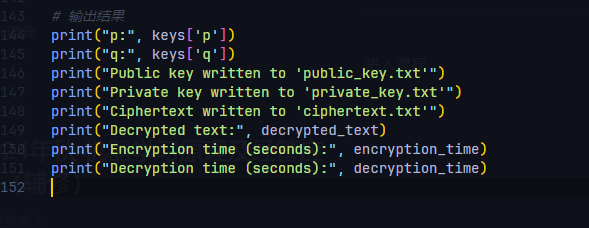


分组加密和解密：



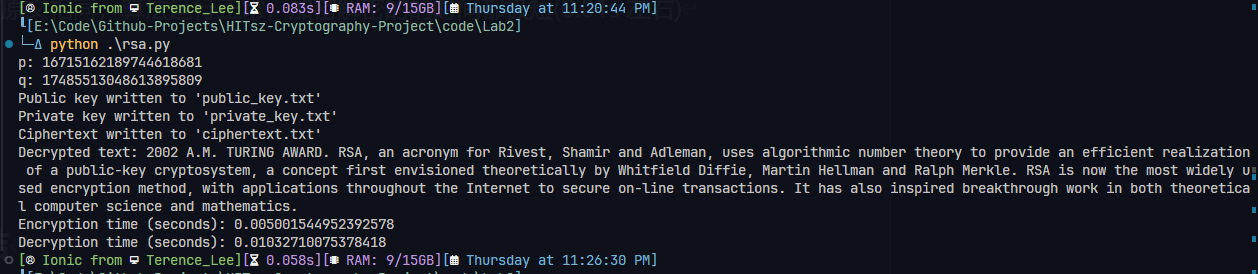
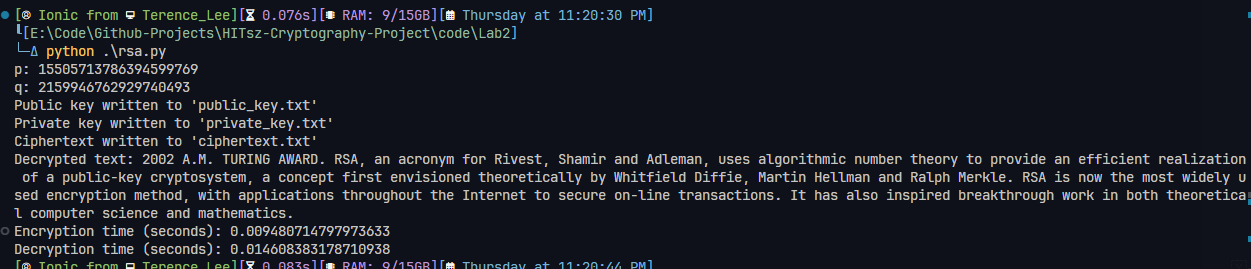
结果输出部分：





# 二、实验结果与分析

程序正确运行的结果截图，包括𝑝, 𝑞, 𝑛,𝑒, 𝑑,𝜙(𝑛)这些参数的值，可输出加密解密时间作为时长参考。



可见，两次实验的p, q值均在64bit数字的范围，加密的结果是Ciphertext， 解密结果与原文相同，算法运行正常；加密解密的的时间都极短(解密0.01s左右)

其中，压缩包内附有密文，可查看

# 三、总结

1、实验过程中遇到的问题有哪些？你是怎么解决的？

1）一开始扩展欧几里得算法的迭代写反了，后续通过单元测试调初始值和数的顺序解决；

2）k较低(一开始设置的是3)存在解密崩掉成乱码的可能性，其原因可能是p, q不是素数但是经过了is\_prime的考验，后续增加k的轮次再也没出现过；

2、关于本实验的意见或建议。

建议加入线上测试和验收平台，就像MIT/Stanford实验那样