**西安电子科技大学网信院**

**信息安全基础与密码学**

**综合实验**

**实 验 报 告（三）**

**基于中国剩余定理的秘密共享方案**

**班级：2118039**

**姓名：**

**学号：**

**日期：2023.12.01**

一、实验目的

1. 实验环境
   1. Windows11
   2. Python3.12
2. 实验目标
   1. 编程实现基于CRT的秘密共享方案
   2. 加深对于中国剩余定理的理解与运用
   3. 体会密码学与数论的紧密联系,将数论的知识运用于密码学的方案设计中
   4. 提高逻辑思维能力与实践能力

二、方案设计

1. 背景
   1. 1983年，Asmuth和Bloom提出了基于中国剩余定理的(t,n)门限秘密共享方案
2. 原理
   1. (t,n)门限，一个秘密k，被分割成n个子秘密(di,ki)，t个子秘密可以恢复出秘密，t-1个子秘密则不能

对于某个秘密k,计算

则子秘密为(di,ki)

* 1. 其中:
     1. ,
     2. 两两互素
     3. N=,M= ，N>M
     4. N>k>M

1. 算法步骤
   1. 读入秘密k
   2. 生成符合的n个di
      1. 列表中随机添加一个数
      2. 增加新的数并判断列表元素是否满足互素，互素则保留
      3. 判断列表中是否有了n个元素,并排序
      4. 计算N,M判断是否符合N>k>M
      5. 符合则返回d
   3. 计算ki≡k(mod di)
   4. 任选t组子秘密，利用中国剩余定理进行恢复
   5. 比较恢复的秘密与原秘密k是否相同

三、方案实现

1. 流程图

图示

描述已自动生成

1. 主要函数介绍
   1. 生成d
   2. 这里我们设定了随机数的生成范围是0到10的200次方，能够在很短的时间内计算出结果。
      1. 三个参数，k, t, n
      2. 使用列表primes存储d
      3. 列表中随机生成一个数
      4. 判断列表中的元素是否为n个，否则进入v，是则进入vii
      5. 列表中增加一个随机数
      6. 判断列表中的元素是否满足互素，不满足，则删除新增加的随机数，重新进入v，满足保留随机数，进入iv
      7. 对列表进行排序
      8. 计算N和M
      9. 判断是否满足N>k>M,满足则返回列表pries，不满足则回到iii

def generate\_d(*k*, *t*, *n*):  
 max = pow(10, 200)  
 while True:  
 primes = [random.randint(2, max)]  
 while len(primes) < *n*:  
 primes.append(random.randint(2, max))  
 if not coprime(primes):  
 primes.pop()  
 sorted\_primes = sorted(primes)  
 product\_n = 1  
 product\_m = 1  
 for num in sorted\_primes[:*t*]:  
 product\_n \*= num  
 for num in sorted\_primes[-*t* + 1:]:  
 product\_m \*= num  
 if product\_n > *k* > product\_m:  
 for i in range(*n*):  
 print(f"d{i + 1}={sorted\_primes[i]}")  
 print(f"N={product\_n},\nM={product\_m}")  
 break  
 return sorted\_primes

* 1. 随机选择t个d

这里，我们随机选择t个密钥对，组成同余方程组

def select\_elements(*input\_list*, *m*):  
 # 确保 m 不大于列表的长度，以避免错误  
 m = min(*m*, len(*input\_list*))  
  
 # 从输入列表中随机选择 m 个元素  
 selected\_elements = random.sample(*input\_list*, *m*)  
  
 return selected\_elements

* 1. CRT求解同余方程中用到的函数，分别是求模逆元、求最大公约数、判断列表元素是否满足两两互素、计算M和Mi。

def mod\_inverse(*a*, *m*):  
 return pow(*a*, -1, *m*)  
  
  
def gcd(*a*, *b*):  
 while *b*:  
 a, b = *b*, *a* % *b* return *a*def coprime(*x*):  
 for i in range(len(*x*)):  
 for j in range(i + 1, len(*x*)):  
 if gcd(*x*[i], *x*[j]) != 1:  
 return False  
 return True  
  
  
def product(*x*):  
 result = 1  
 M = []  
 for i in *x*:  
 result \*= i  
 for i in range(len(*x*)):  
 temp = 1  
 for j in range(len(*x*)):  
 if i != j:  
 temp \*= *x*[j]  
 M.append(temp)  
 return result, M

* 1. CRT的主要代码

def crt(*data*):  
 num\_a = len(*data*) // 2  
 ai = *data*[:num\_a]  
 mi = *data*[num\_a:]  
 # 计算m  
 m, Mj = product(mi)  
 # 计算Mj的逆元  
 inv\_Mj = [mod\_inverse(Mj[i], mi[i]) for i in range(len(Mj))]  
 # 计算x  
 x = [ai[i] \* Mj[i] \* inv\_Mj[i] for i in range(len(ai))]  
 # 求解  
 return sum(x) % m

* 1. 主函数
     1. 设定t和n的值
     2. 遍历读入所有文件
     3. 调用其他函数

if \_\_name\_\_ == "\_\_main\_\_":  
 t = 3  
 n = 5  
 folder\_path = r"test\_data"  
 files = os.listdir(folder\_path)  
 for file\_name in files:  
 file\_path = os.path.join(folder\_path, file\_name)  
 print(file\_name)  
 with open(file\_path, 'r') as f:  
 k = int(f.read())  
 raw\_data = generate\_d(k, t, n)  
 selected\_data = select\_elements(raw\_data, t)  
 data = []  
 for i in range(t):  
 data.append(k % selected\_data[i])  
 data += selected\_data  
 result = crt(data)  
 print(f"k={k}")  
 print(f"result={result}")  
 if result == k:  
 print("恢复成功")  
 else:  
 print("恢复失败")  
 print()

1. 算法的主要代码

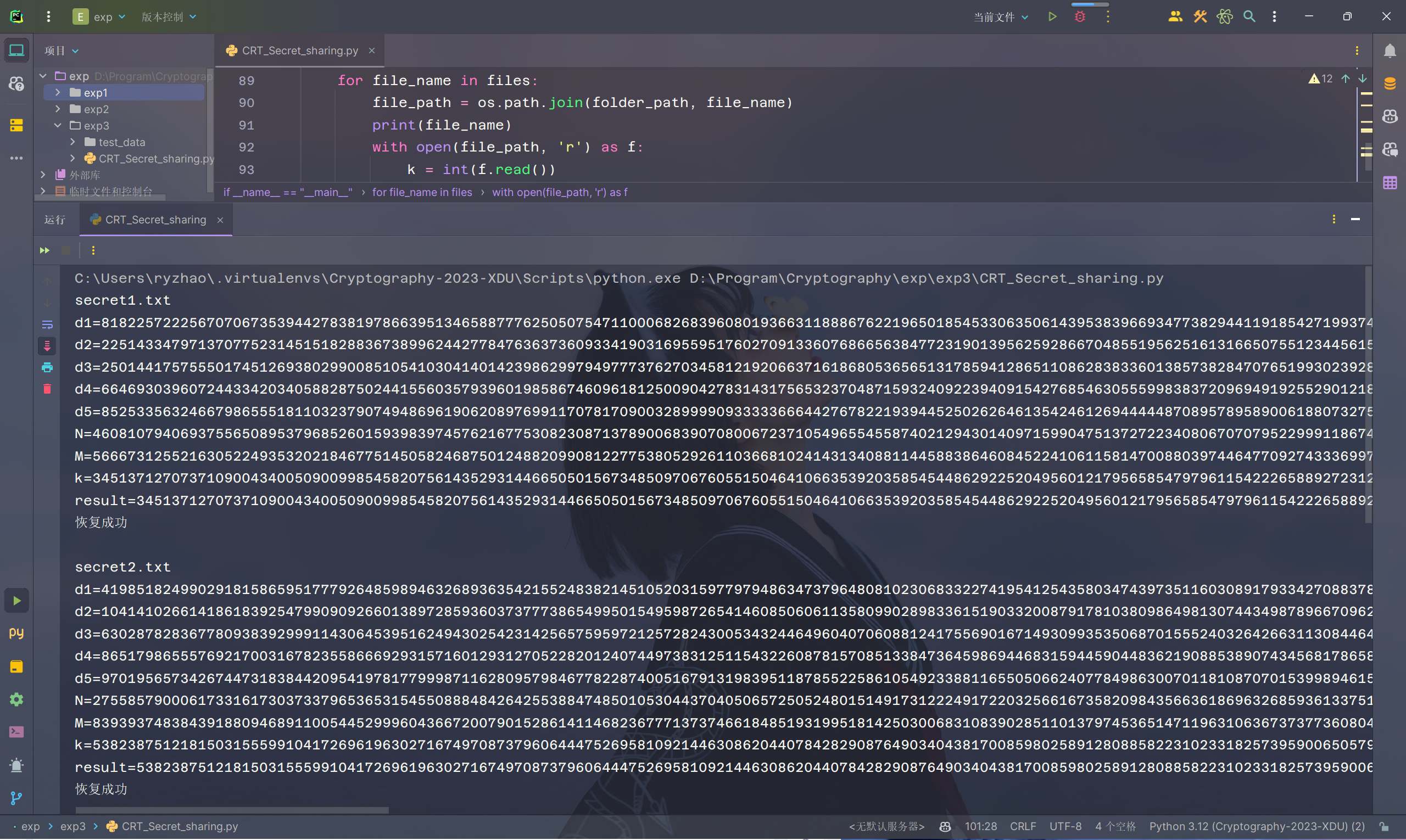
import os  
import random  
  
  
def generate\_d(*k*, *t*, *n*):  
 max = pow(10, 200)  
 while True:  
 primes = [random.randint(2, max)]  
 while len(primes) < *n*:  
 primes.append(random.randint(2, max))  
 if not coprime(primes):  
 primes.pop()  
 sorted\_primes = sorted(primes)  
 product\_n = 1  
 product\_m = 1  
 for num in sorted\_primes[:*t*]:  
 product\_n \*= num  
 for num in sorted\_primes[-*t* + 1:]:  
 product\_m \*= num  
 if product\_n > *k* > product\_m:  
 for i in range(*n*):  
 print(f"d{i + 1}={sorted\_primes[i]}")  
 print(f"N={product\_n},\nM={product\_m}")  
 break  
 return sorted\_primes  
  
  
def mod\_inverse(*a*, *m*):  
 return pow(*a*, -1, *m*)  
  
  
def gcd(*a*, *b*):  
 while *b*:  
 a, b = *b*, *a* % *b* return *a*def coprime(*x*):  
 for i in range(len(*x*)):  
 for j in range(i + 1, len(*x*)):  
 if gcd(*x*[i], *x*[j]) != 1:  
 return False  
 return True  
  
  
def product(*x*):  
 result = 1  
 M = []  
 for i in *x*:  
 result \*= i  
 for i in range(len(*x*)):  
 temp = 1  
 for j in range(len(*x*)):  
 if i != j:  
 temp \*= *x*[j]  
 M.append(temp)  
 return result, M  
  
  
def select\_elements(*input\_list*, *m*):  
 # 确保 m 不大于列表的长度，以避免错误  
 m = min(*m*, len(*input\_list*))  
  
 # 从输入列表中随机选择 m 个元素  
 selected\_elements = random.sample(*input\_list*, *m*)  
  
 return selected\_elements  
  
  
def crt(*data*):  
 num\_a = len(*data*) // 2  
 ai = *data*[:num\_a]  
 mi = *data*[num\_a:]  
 # 计算m  
 m, Mj = product(mi)  
 # 计算Mj的逆元  
 inv\_Mj = [mod\_inverse(Mj[i], mi[i]) for i in range(len(Mj))]  
 # 计算x  
 x = [ai[i] \* Mj[i] \* inv\_Mj[i] for i in range(len(ai))]  
 # 求解  
 return sum(x) % m  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 t = 3  
 n = 5  
 folder\_path = r"test\_data"  
 files = os.listdir(folder\_path)  
 for file\_name in files:  
 file\_path = os.path.join(folder\_path, file\_name)  
 print(file\_name)  
 with open(file\_path, 'r') as f:  
 k = int(f.read())  
 raw\_data = generate\_d(k, t, n)  
 selected\_data = select\_elements(raw\_data, t)  
 data = []  
 for i in range(t):  
 data.append(k % selected\_data[i])  
 data += selected\_data  
 result = crt(data)  
 print(f"k={k}")  
 print(f"result={result}")  
 if result == k:  
 print("恢复成功")  
 else:  
 print("恢复失败")  
 print()

四、数据分析

读入测试数据文件夹中**全部的测试数据**中的秘密k

输出格式为：

1. 读入的文件名称
2. d1,d2,d3,d4,d5的值
3. N的值
4. M的值
5. 秘密k
6. 恢复的结果
7. 是否恢复成功



文本

描述已自动生成

五、思考与总结

1. 在基于中国剩余定理的(*t*, *n*)秘密共享方案中，少于*t*个子秘密，是否能够正确恢复出秘密？请简述原因。

不能正确恢复出秘密。

当我们任意选择t-1个子秘密(dj1,kj1), (dj2,kj2),… , (djt-1,kjt-1)时，

通过中国剩余定理，我们可以求解出,其中M1= dj1dj2…djt-1.

但是根据秘密分割时的条件，我们知道N>k>M,其中N是t个从小到大d的乘积，M是t-1个从大到小d的乘积。

因此会有N>k>M>M1,因而没有足够的信息去正确恢复出秘密。不过，仍带有部分k的信息。

1. 实验过程中还遇到了什么问题，如何解决的？通过该实验有何收获？

找出符合的d这个条件非常难以下手。

我们没有办法给出di的范围，秘密k太大了，十进制500位，如果随机从10的501次方内生成随机数查看是否满足条件的话，会非常缓慢，我们设定了范围在10的200次方以内，很好的解决了问题。

另外，我们可以使用smypy库直接生成一个区间内的大素数，这样就省去了判断两两互素的过程。