**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：2118021**

**姓名：**

**学号：**

**日期：2023年11月5日**

一、实验目的

1. 实验环境

macOS Sonoma、Python 3.9

1. 实现目标

能够根据秘密共享方案的要求，对大整数实现基于中国剩余定理的（t, n）门限的秘密共享，并能够通过中国剩余定理测试选择不同d的个数时，能否恢复原秘密。

二、方案设计

1. 背景

对于存储的秘密，通常既不希望其保存过于集中，使得一个参与者就能够恢复秘密，也不希望秘密保存过于分散，要使得一些参与者出现异常时，也能够正确恢复秘密。

因此在中国剩余定理的基础上，提出了秘密共享方案，即将秘密拆分为多个部分，只有存在不少于一定个数的参与者时，秘密才能够正确恢复。

1. 原理

对于一个秘密，可以将其分割为n个子秘密，同时要求当有不少于t个子秘密时，能够将原秘密恢复。即对于某个秘密k，将其分割为n个子秘密，则原秘密被分割为。

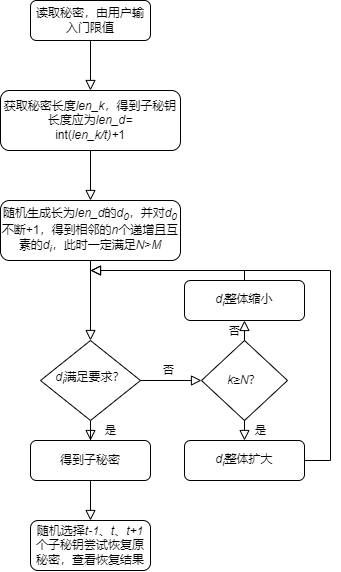
分割的参数应满足

1. 严格递增；
2. 两两互素；
3. 令,有；

若要恢复原秘密，从中选取个子秘密计算即可得到原秘密。

三、方案实现

1. 算法流程图



1. 主要函数的介绍

* **def initialise\_d(n, t, d, k):**

根据秘密长度与门限初始化。

* **def calculate\_proper\_d(d, n, t, k):**

寻找符合要求的

* **def chinese\_remainder\_theorem(eq) -> [int, int]:**

用中国剩余定理恢复原秘密

1. 算法实现的主要代码
2. import random
3. from rsa.common import inverse
4. class Equation:
5. a = 0
6. m = 0
7. def \_\_init\_\_(self, a, m):
8. self.a = a
9. self.m = m
10. def main():
11. k = []
12. with open("secret1.txt") as secret1:
13. for line in secret1:
14. k.append(int(line))
15. secret1.close()
16. with open("secret2.txt") as secret2:
17. for line in secret2:
18. k.append(int(line))
19. secret2.close()
20. n = int(input("请输入n:\n"))
21. t = int(input("请输入t:\n"))
22. # 分别对两个文件中的秘密进行秘密共享处理
23. for i in range(2):
24. d = []
25. initialise\_d(n, t, d, k[i])
26. [N, M] = calculate\_proper\_d(d, n, t, k[i])
27. print("第" + str(i + 1) + "个秘密的d分别为：")
28. for j in range(len(d)):
29. print(d[j])
30. print("第" + str(i + 1) + "个秘密的N为：")
31. print(N)
32. print("第" + str(i + 1) + "个秘密的M为：")
33. print(M)
34. print()
35. # 分别测试选择t-1、t、t+1个d时能否正确得到秘密
36. d\_num = [t - 1, t, t + 1]
37. for test\_round in range(3):
38. eq = []
39. chosen\_d = set()
40. # 随机选择d
41. while len(chosen\_d) < d\_num[test\_round]:
42. chosen\_d.add(random.randint(0, n - 1))
43. list\_chosen\_d = list(chosen\_d)
44. # 针对选择的d使用中国剩余定理反求原秘密
45. for j in list\_chosen\_d:
46. eq.append(Equation(k[i] % d[j], d[j]))
47. result = chinese\_remainder\_theorem(eq)
48. print("使用", end="")
49. for index in range(len(list\_chosen\_d)):
50. print("d" + str(list\_chosen\_d[index]), end="")
51. if index < len(list\_chosen\_d) - 1:
52. print("、", end="")
53. print("恢复秘密为：")
54. print(result[0])
55. if result[0] - k[i] == 0:
56. print("恢复的秘密与原秘密相同")
57. else:
58. print("恢复的秘密与原秘密不同")
59. print()
60. print()
61. def initialise\_d(n, t, d, k):
62. """
63. 初始化di
64. """
65. len\_k = len(str(k))
66. # 根据秘密大小生成相应位数的d
67. d.append(random.randint(10 \*\* (int(len\_k / t) + 1), 10 \*\* (int(len\_k / t) + 2)))
68. while len(d) < n:
69. d.append(d[len(d) - 1] + 1)
70. while not is\_relatively\_prime(d):
71. d[len(d) - 1] = d[len(d) - 1] + 1
72. def calculate\_proper\_d(d, n, t, k):
73. """
74. 寻找符合要求的d
75. """
76. N = 1
77. for i in range(t):
78. N = N \* d[i]
79. M = 1
80. for i in range(n - 1, n - t, -1):
81. M = M \* d[i]
82. # 不满足要求则持续更改d
83. while not (N > k > M):
84. if not N > M:
85. d.clear()
86. initialise\_d(n, t, d, k)
87. elif k >= N:
88. # k >= N时，选择的d整体增大
89. for i in range(len(d) - 1):
90. d[i] = d[i + 1]
91. while not is\_relatively\_prime(d):
92. d[len(d) - 1] = d[len(d) - 1] + 1
93. else:
94. # k <= M时，选择的d整体减小
95. for i in range(len(d) - 1, 0, -1):
96. d[i] = d[i - 1]
97. while not is\_relatively\_prime(d):
98. d[0] = d[0] - 1
99. N = 1
100. for i in range(t):
101. N = N \* d[i]
102. M = 1
103. for i in range(n - 1, n - t, -1):
104. M = M \* d[i]
105. return [N, M]
106. def chinese\_remainder\_theorem(eq) -> [int, int]:
107. """
108. 中国剩余定理
109. return [x, m]
110. """
111. # 中国剩余定理求解
112. M = 1
113. x = 0
114. for i in range(len(eq)):
115. M = M \* eq[i].m
116. for i in range(len(eq)):
117. mi = 1
118. for j in range(len(eq)):
119. if j != i:
120. mi = mi \* eq[j].m
121. x = x + mi \* inverse(mi, eq[i].m) \* eq[i].a
122. x = x % M
123. return [x, M]
124. def modular\_exponentiation(a, p, m):
125. """
126. 快速模指数运算，a^p(mod m)
127. """
128. binary\_p = bin(p)
129. reversed\_binary\_p = binary\_p[len(binary\_p):1:-1]
130. # 运算结果
131. result = 1
132. # 每一个二进制位对应底数
133. bn = a
134. for n in range(len(reversed\_binary\_p)):
135. result = result \* bn \*\* int(reversed\_binary\_p[n]) % m
136. bn = bn \*\* 2 % m
137. return result
138. def gcd(a, b):
139. """
140. 用欧几里得算法计算最大公因数
141. """
142. if a < b:
143. temp = a
144. a = b
145. b = temp
146. if a % b == 0:
147. return b
148. return gcd(b, a % b)
149. def is\_relatively\_prime(d):
150. """
151. 判断di是否两两互素
152. 返回True表示两两互素，否则返回False
153. """
154. for i in range(len(d)):
155. for j in range(i + 1, len(d)):
156. if gcd(d[i], d[j]) != 1:
157. return False
158. return True
159. if \_\_name\_\_ == "\_\_main\_\_":
160. main()

四、数据分析

1. 算法测试数据的分析

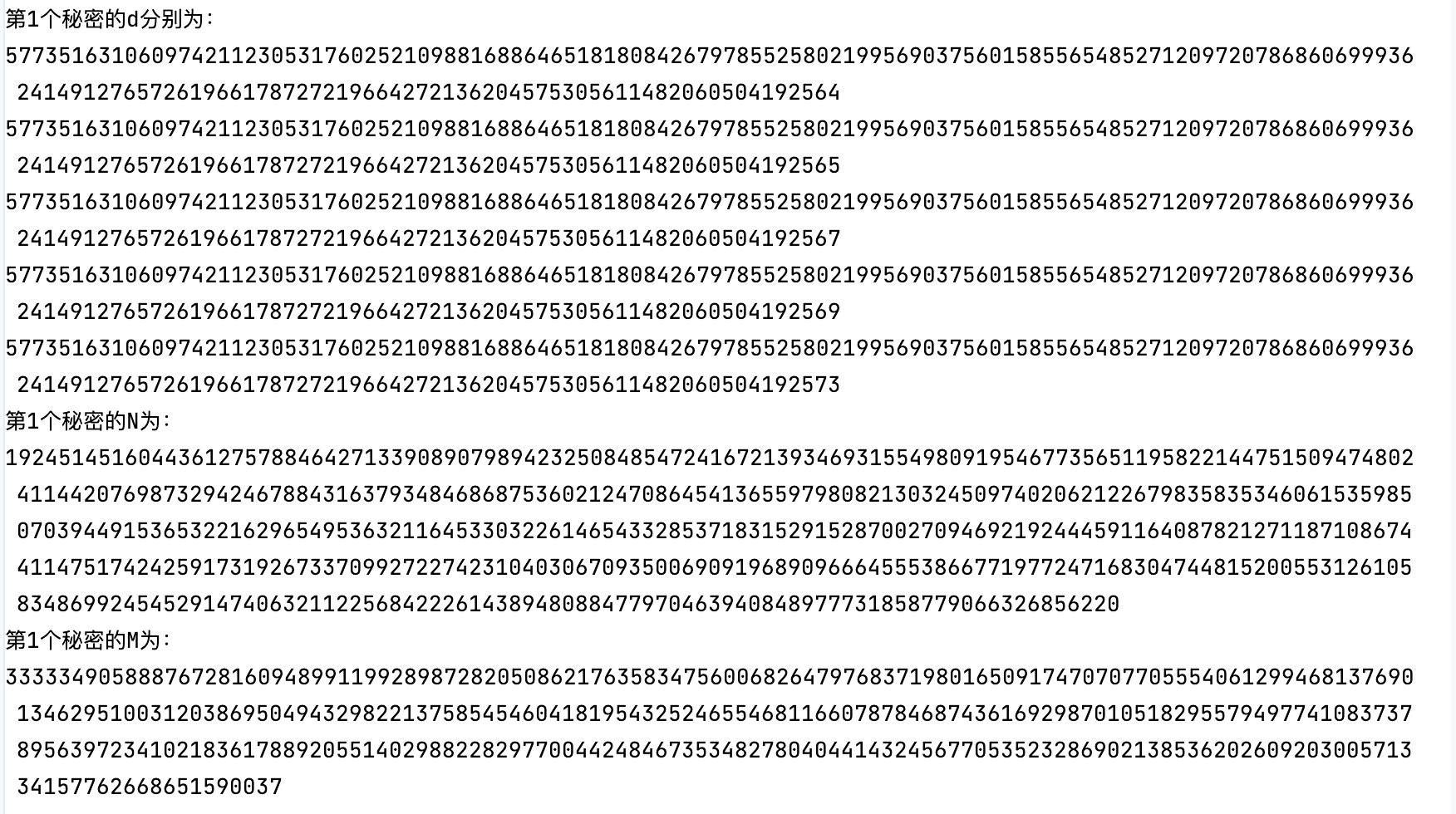
本题使用第一个秘密共享，共500位。由于门限为（3,5），此时若要满足共享条件，根据条件的上限选择子秘钥长度，即的长度应满足，且只有选择不少于3个子秘密才能恢复原秘密。

1. 运行结果截图

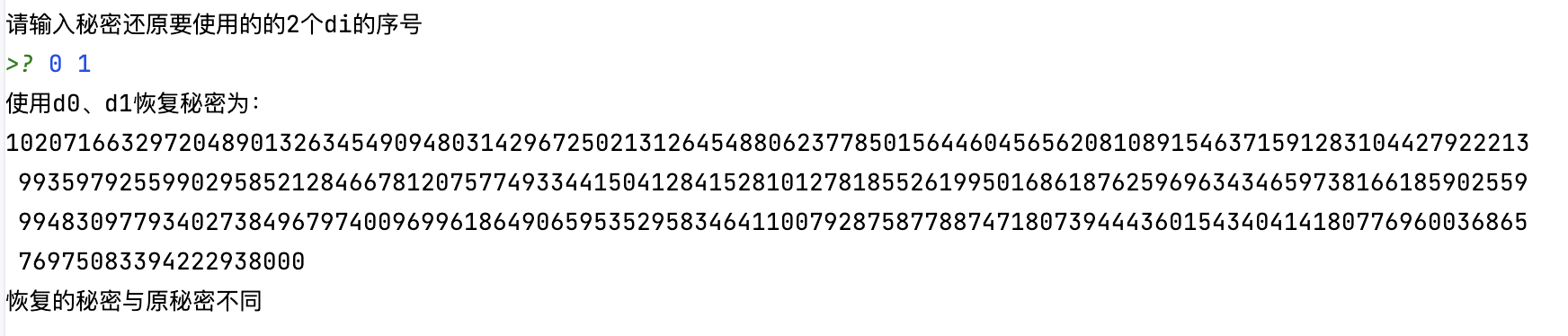
手动输入n和t：



secret1.txt的d、m、n情况：

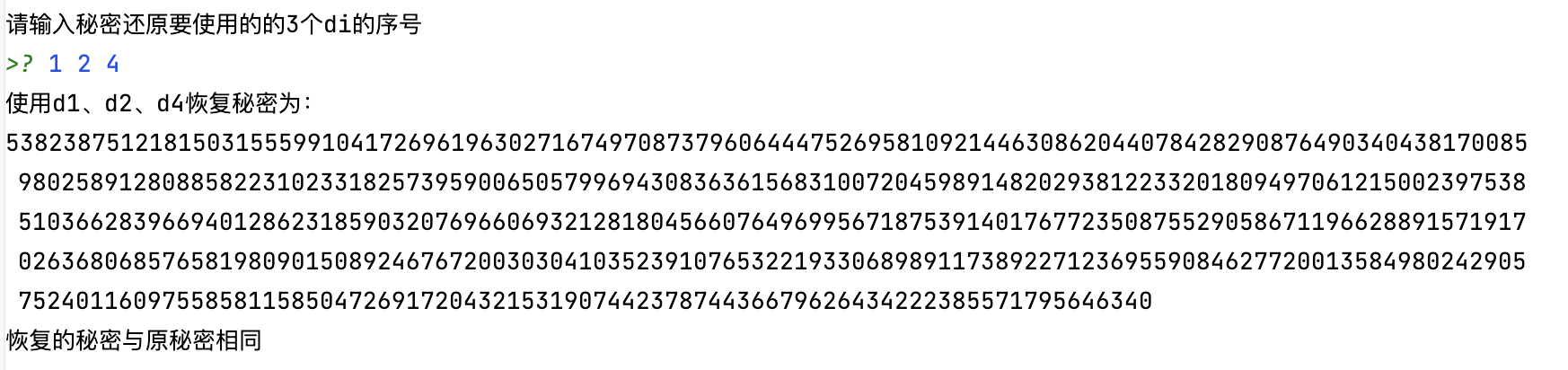


之后分别使用t-1、t、t+1个子秘密来进行秘密还原，首先指定t-1即2个秘密，此处使用d0和d1:



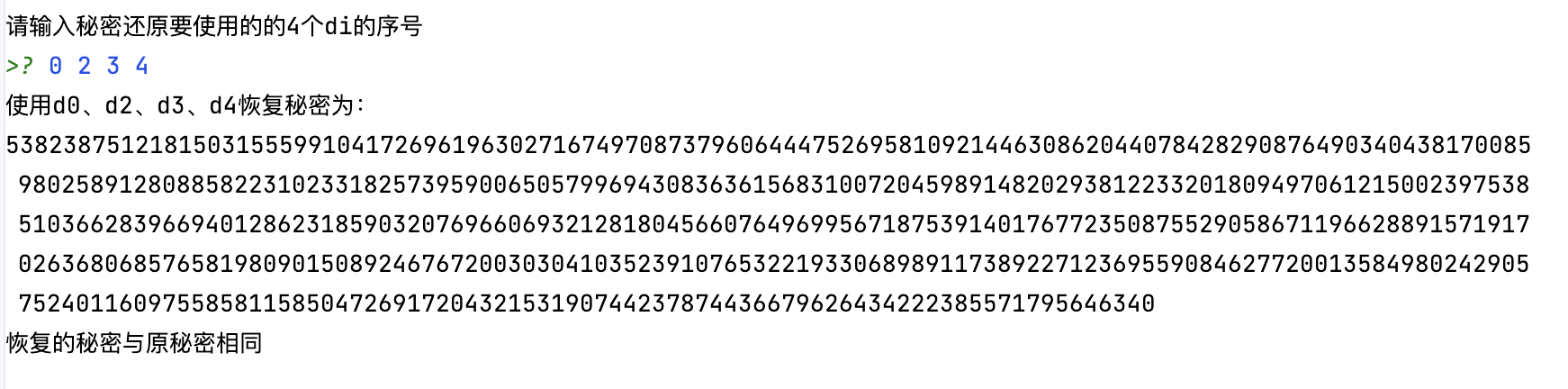
可以看到，当选择2个子秘密时无法复原秘密。

再选择使用d1、d2、d4三个子秘密来尝试复原：



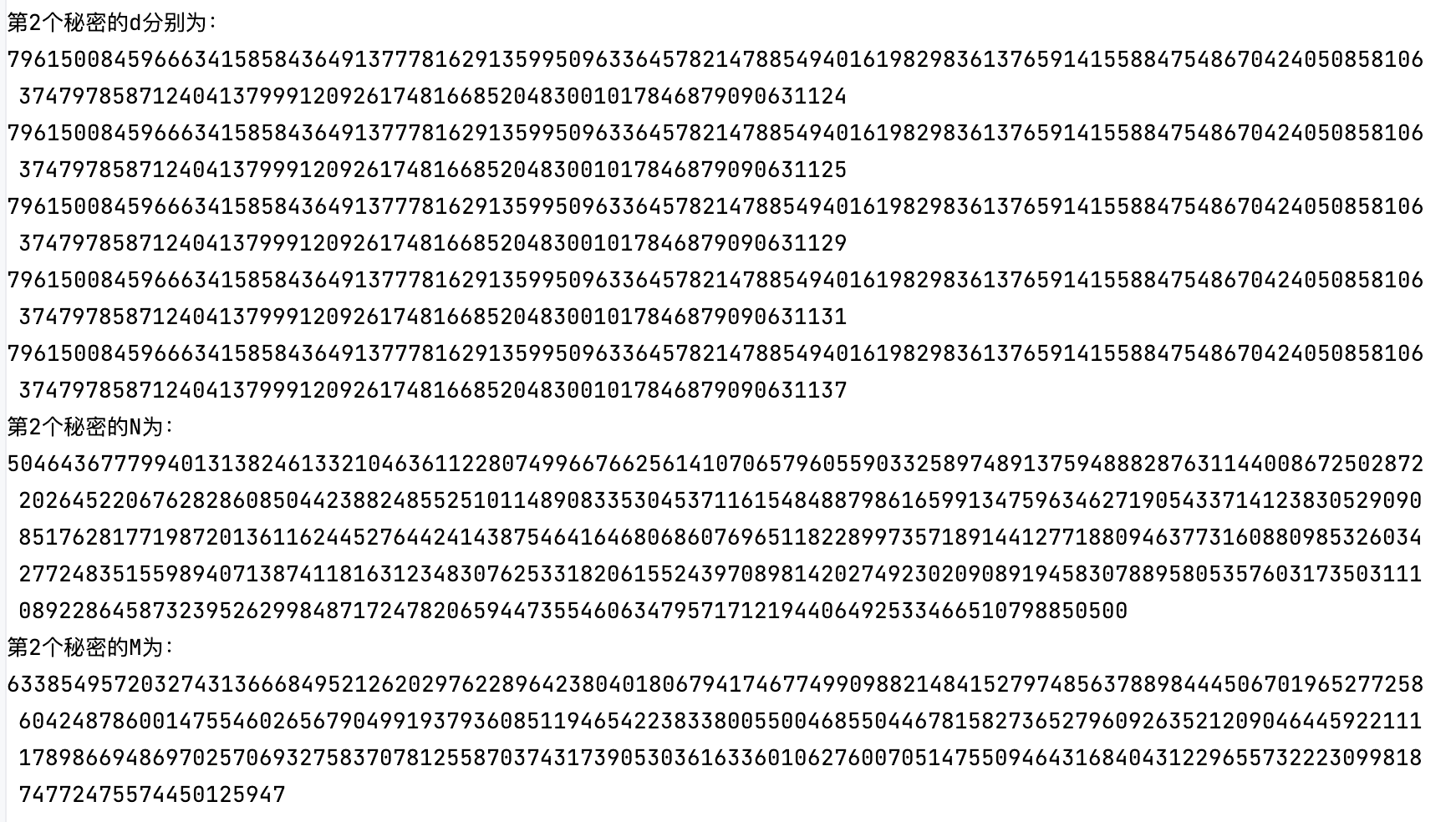
可以看到，当选择3个子秘密时可以复原秘密。

最后选择使用d0、d2、d3、d4四个子秘密来尝试复原：

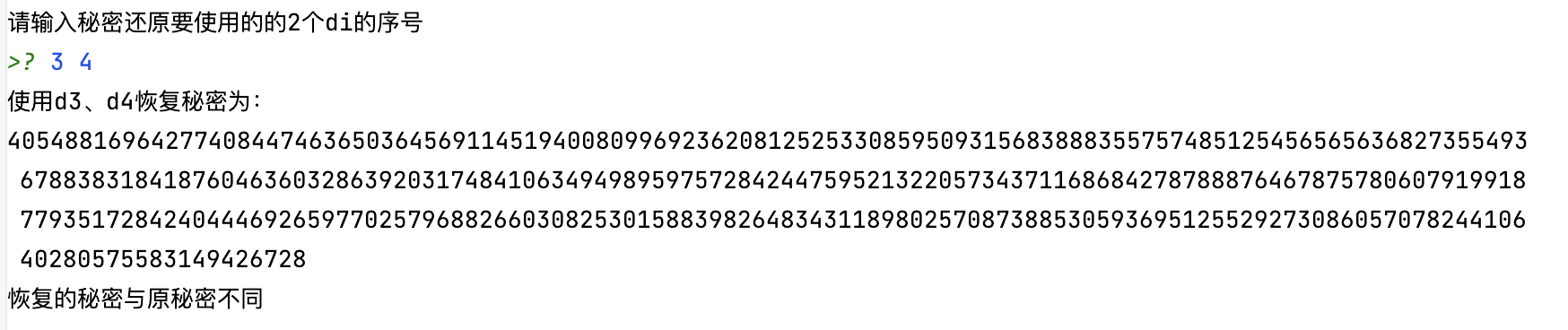


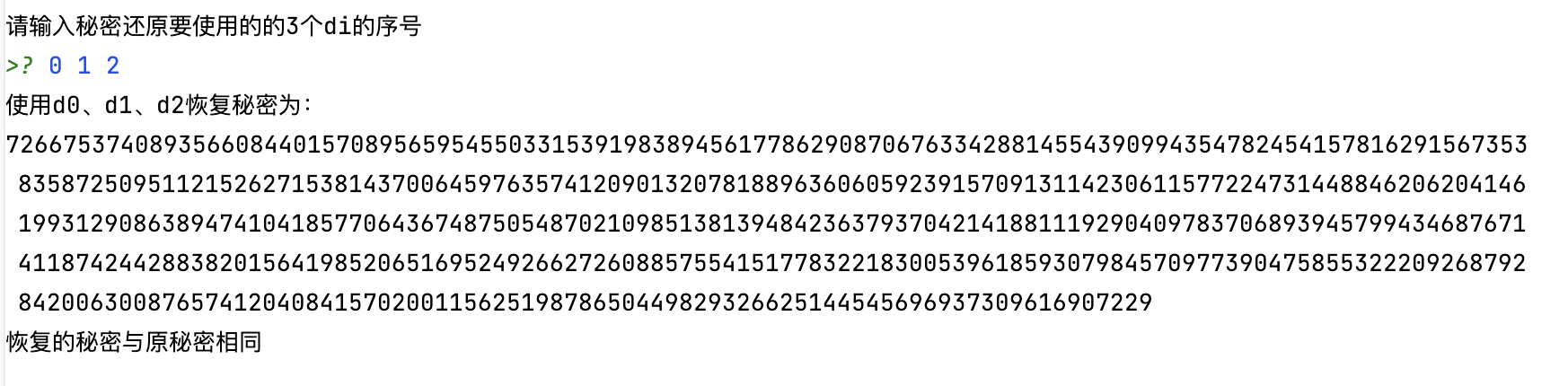
可以看到，当选择4个子秘密时可以复原秘密。

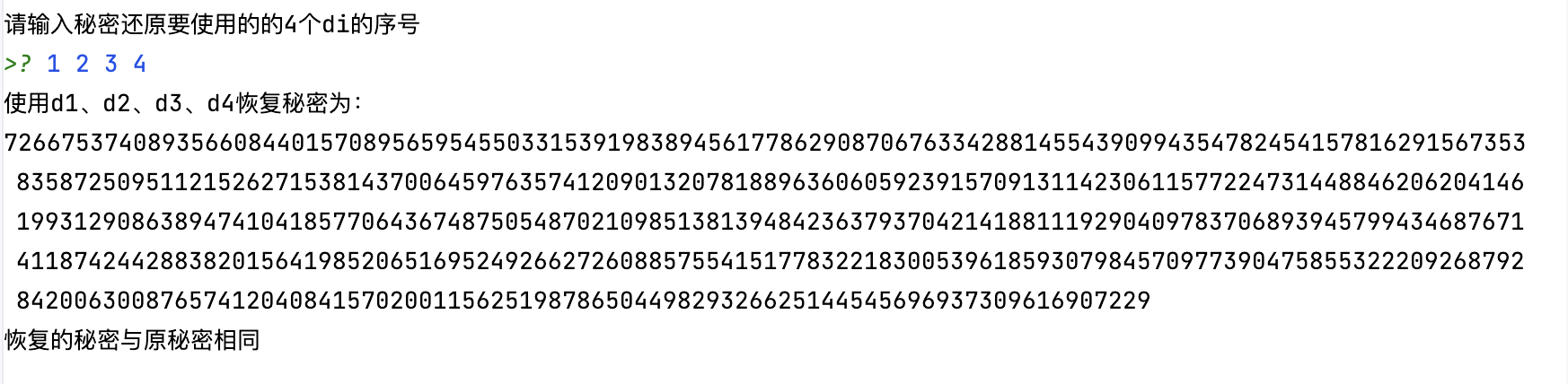
secret2.txt的d、m、n情况：



如secret1.txt一样，此处分别选择t-1、t、t+1个di尝试对secret2.txt的秘密进行复原，结果截图如下：







可以看到，当选择2个子秘密时无法复原秘密，而选择3、4个子秘密时可以恢复。

综上所述，当选择少于个子秘密时，无法恢复原秘密，而选择至少个子秘密时可以恢复。

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。

当选择少于个子秘密时，通过中国剩余定理得到的秘密为，且，即恢复出的秘密仅为原秘密的一部分，不能恢复出原秘密。

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？

实验时选择子秘钥的长度存在一定困难。当子秘钥长度选择不合适时，会使得程序对其调整较长时间才能获得最终结果。由于选择连续个递增且互素的数，一定满足条件(1)至(3)，此时仅满足条件(4)即可。此时若根据下限确定子秘钥长度为，所得通常偏大，程序需进行较长时间调整才能满足该要求，而当根据上限确定子秘钥长度时，可较快得到合适的子秘钥。