**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（五）**

**SM2椭圆曲线公钥密码算法**

**班级：2118021**

**姓名：**

**学号：**

**日期：2023年11月18日**

一、实验目的

1. 实验环境

macOS Sonoma、Python 3.9

1. 实现目标

编程实现SM2椭圆曲线公钥密码算法。

能够对输入的明文消息使用SM2公钥密码算法正确加密，生成密文，并能够通过解密算法正确恢复消息内容。

二、方案设计

1. 背景

SM2算法是国家密码管理局于2010年12月17日发布的国密标准椭圆曲线加密算法。对于一般椭圆曲线的离散对数问题，目前只存在指数级计算复杂度的求解方法。与大数分解问题及有限域上离散对数问题相比，椭圆曲线离散对数问题的求解难度要大得多。因此，在相同安全程度要求下，椭圆曲线密码较其他公钥密码所需的秘钥规模要小得多。

SM2算法和RSA算法都是公钥密码算法，SM2算法是一种更先进安全的算法，在我们国家商用密码体系中被用来替换RSA算法。

1. 原理

SM2公钥密码算法是基于椭圆曲线倍点问题而设计，即已知一个点与其倍点的倍数，易求得这一倍点的坐标。但是给定一个点与其倍点，难以求得倍点相对的倍数。

定义在素域Fp（p是大于3的素数）上的椭圆曲线方程为：

， 且 

椭圆曲线E(Fp)定义为：

，其中是无穷远点。

椭圆曲线E(Fp)上的点按照以下加法运算规则，构成一个交换群：

1. ;
2. ;
3. ；
4. 两个非互逆的不同点相加的规则：

设， 且 ，

设，

则，

其中。

1. 倍点规则：

设，且，

设，

则，

其中

1. 算法步骤
   1. 确定椭圆曲线

首先参照国家密码局所给公钥加密算法的示例，创建椭圆曲线方程。本次实验使用附录A.2中示例1在素域上的椭圆曲线的参数。

随机生成不小于1不大于n-1的作为私钥，计算作为公钥。

* 1. 加密

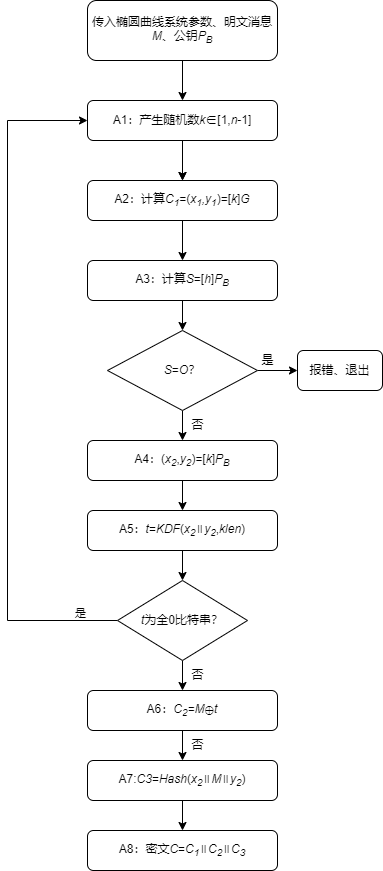
参考国家密码局所给加密算法进行加密

1. 随机产生不小于1不大于的；
2. 计算，并将转换为比特串；
3. 计算，判断是否是无穷远点，若是则报错、退出；
4. 计算，并将其转换为比特串
5. 计算，判断是否全为0，若是则说明取值不合适，返回步骤1)重新选取；
6. 计算，即对消息加密；
7. 计算，用于解密时校验；
8. 输出密文。
   1. 解密
9. 从密文中取出，将其传唤为椭圆曲线上的点，并验证该点是否在给定椭圆曲线上，若不在则报错、退出；
10. 计算，判断是否是无穷远点，若是则报错、退出；
11. 计算，并将该点坐标转换为比特串；
12. 计算，判断是否全为0，若是则报错、退出；
13. 从密文中取出，计算；
14. 计算，从密文中取出，比较，与是否相等，用于校验解密是否正确，若相等的说明解密正确，否则报错、退出；
15. 输出明文。

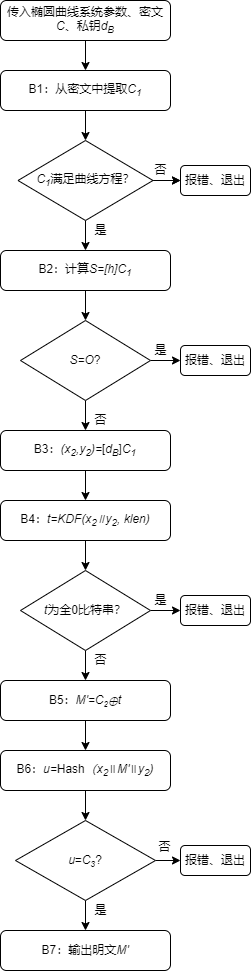
三、方案实现

1. 算法流程图

加密算法：



解密算法：



1. 主要函数的介绍

def encrypt(m, xG, yG, xB, yB, a, p, n):

加密算法，传入明文、公钥及必要参数，返回密文

def decrypt(C, a, b, p, dB, point\_len):

解密算法，传入密文、私钥及必要参数，返回明文

def add\_points(x1, y1, x2, y2, a, p):

计算椭圆曲线上两点相加的和

def multiple\_point(x1, y1, k, a, p):

计算倍点，使用第一部分附录A.3.2中算法一：二进制展开法快速计算倍点

def kdf(z, klen):

kdf函数

def Hv(str):

密码杂凑函数SM3，输入输出均为16进制

1. 算法实现的主要代码
2. from random import randint
3. from gmssl import sm3, func
4. from rsa.common import inverse
5. def add\_points(x1, y1, x2, y2, a, p):
6. """
7. 椭圆曲线上两点相加
8. """
9. if x1 == x2 and y1 + y2 == p:
10. return False
11. elif x1 == x2 and y1 == y2:
12. lambda\_of\_points = ((3 \* x1 \*\* 2 + a) \* inverse(2 \* y1, p)) % p
13. else:
14. lambda\_of\_points = ((y2 - y1) \* inverse(x2 - x1, p)) % p
15. x3 = (lambda\_of\_points \*\* 2 - x1 - x2) % p
16. y3 = (lambda\_of\_points \* (x1 - x3) - y1) % p
17. return x3, y3
18. def multiple\_point(x1, y1, k, a, p):
19. """
20. 计算倍点，使用第一部分附录A.3.2中算法一：二进制展开法快速计算倍点
21. """
22. k = bin(k)[2:]
23. qx, qy = x1, y1
24. for i in range(1, len(k)):
25. qx, qy = add\_points(qx, qy, qx, qy, a, p)
26. if k[i] == '1':
27. qx, qy = add\_points(qx, qy, x1, y1, a, p)
28. return qx, qy
29. def str2byte(str):
30. """
31. 字符串转换为01比特串
32. """
33. rule = 'utf-8'
34. byte\_str = str.encode(rule)
35. bc = [bin(i)[2:].rjust(8, '0') for i in byte\_str]
36. bytes = ''.join(bc)
37. return bytes
38. def kdf(z, klen):
39. """
40. kdf函数
41. """
42. ct = 1
43. v = 256
44. k = ''
45. for i in range(int(klen / v) + 1):
46. k = k + Hv(hex(int(z + '{:02x}'.format(ct), 16))[2:])
47. ct = ct + 1
48. bin\_k = bin(int(k, 16))[2:]
49. return hex(int(bin\_k[:klen], 2))[2:]
50. def Hv(str):
51. """
52. 密码杂凑函数SM3，输入输出均为16进制
53. """
54. return sm3.sm3\_hash(func.bytes\_to\_list(bytes(str, encoding='utf-8')))
55. def encrypt(m, xG, yG, xB, yB, a, p, n):
56. """
57. 加密算法
58. """
59. # 首先将消息转换为二进制比特串
60. byte\_m = str2byte(m)
61. while 1:
62. # A1：生成随机的k
63. k = randint(1, n - 1)
64. # k = 0x4C62EEFD6ECFC2B95B92FD6C3D9575148AFA17425546D49018E5388D49DD7B4F
65. # A2：计算C1并转换为16进制比特串
66. c1x, c1y = multiple\_point(xG, yG, k, a, p)
67. c1x = hex(c1x)[2:]
68. c1y = hex(c1y)[2:]
69. C1 = '04' + c1x + c1y
70. print("C1:\n"+C1+"\n")
71. # A3：根据O + O = O判断S是否是无穷远点
72. h = 1
73. hPBx, hPBy = multiple\_point(xB, yB, h, a, p)
74. hPBx2, hPBy2 = multiple\_point(hPBx, hPBy, 2, a, p)
75. if hPBx == hPBx2 and hPBy == hPBy2:
76. print("加密时S为无穷远点，发生错误")
77. exit()
78. # A4：计算(x2, y2)
79. x2, y2 = multiple\_point(xB, yB, k, a, p)
80. x2 = hex(x2)[2:]
81. y2 = hex(y2)[2:]
82. print("x2:\n"+x2+"\ny2:\n"+y2)
83. # A5：计算t = KDF(x2 || y2, klen)，并根据t是否是全0字符串判断k是否合适
84. t = kdf(x2 + y2, len(byte\_m))
85. if int(t, 16) != 0:
86. break
87. print("t:\n"+t)
88. # A6：计算C2 = M ⊕ t
89. C2 = int(byte\_m, 2) ^ int(t, 16)
90. C2 = str(bin(C2)[2:])
91. C2 = hex(int(C2, 2))[2:]
92. print("C2:\n"+C2)
93. # A7：计算C3 = Hash(x2 || M || y2)
94. C3 = Hv(x2 + hex(int(byte\_m, 2))[2:] + y2)
95. print("C3:\n"+C3)
96. # A8：输出密文
97. return C1 + C2 + C3
98. def decrypt(C, a, b, p, dB, point\_len):
99. """
100. 解密算法
101. """
102. C1 = C[:2 + 2 \* point\_len]
103. C2 = C[2 + 2 \* point\_len:-64]
104. C3 = C[len(C1) + len(C2):]
105. # B1：取C1并判断是否是椭圆曲线上的点
106. C1 = C1[2:]
107. x1 = C1[:int(len(C1) / 2)]
108. y1 = C1[int(len(C1) / 2):]
109. x1 = int(x1, 16)
110. y1 = int(y1, 16)
111. if pow(y1, 2, p) != (pow(x1, 3, p) + a \* x1 + b) % p:
112. print("解密时C1不满足椭圆曲线方程，发生错误")
113. exit()
114. print("x1:\n"+hex(x1)[2:]+"\ny1:\n"+hex(y1)[2:])
115. # B2：根据O + O = O判断S是否是无穷远点
116. h = 1
117. hPBx, hPBy = multiple\_point(x1, y1, h, a, p)
118. hPBx2, hPBy2 = multiple\_point(hPBx, hPBy, 2, a, p)
119. if hPBx == hPBx2 and hPBy == hPBy2:
120. print("解密时S为无穷远点，发生错误")
121. exit()
122. # B3：计算(x2,y2)
123. x2, y2 = multiple\_point(x1, y1, dB, a, p)
124. x2 = hex(x2)[2:]
125. y2 = hex(y2)[2:]
126. print("x2:\n" + x2 + "\ny2:\n" + y2)
127. # B4：计算t = KDF(x2 || y2, klen)，若t是否是全0字符串则报错
128. t = kdf(x2 + y2, len(C2) \* 4)
129. if int(t, 16) == 0:
130. print("解密时t取值发生错误")
131. exit()
132. print("t:\n" + t)
133. # B5：取出C2并计算M′ = C2 ⊕ t
134. bytes\_decrypted\_m = int(C2, 16) ^ int(t, 16)
135. # B6:计算u = Hash(x2 || M′ || y2)，并判断u是否等于C3，若不等于则报错
136. u = Hv(x2 + hex(bytes\_decrypted\_m)[2:] + y2)
137. if u != C3:
138. print("解密时u发生错误")
139. exit()
140. print("u:\n"+u)
141. # B7：输出明文
142. hex\_decrypted\_m = hex(bytes\_decrypted\_m)[2:]
143. m = str(bytes.fromhex(hex\_decrypted\_m))[2:-1]
144. print("解密算法中数据验证通过，恢复的消息为：\n" + m)
145. return m
146. if \_\_name\_\_ == "\_\_main\_\_":
147. # 初始参数设置，使用Fp上椭圆曲线
148. p = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFF
149. a = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFC
150. b = 0x28E9FA9E9D9F5E344D5A9E4BCF6509A7F39789F515AB8F92DDBCBD414D940E93
151. n = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF7203DF6B21C6052B53BBF40939D54123
152. xG = 0x32C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C7
153. yG = 0xBC3736A2F4F6779C59BDCEE36B692153D0A9877CC62A474002DF32E52139F0A0
154. print("本次加密过程中\np为：\n" + hex(p) + "\na为：\n" + hex(a) + "\nb为：\n" + hex(b) +"\nn为：\n" + hex(n) +"\nGx为：\n" + hex(xG) +"\nGy为：\n" + hex(yG))
155. print()
156. # 生成公私钥
157. dB = randint(1, n - 2)
158. print("产生随机数k：",dB)
159. xB, yB = multiple\_point(xG, yG, dB, a, p)
160. print("私钥dB为：\n" + hex(dB) + "\n公钥PB为：\n(" + hex(xB) + ", " + hex(yB) + ")\n")
161. m = input("请输入要加密的消息：\n")
162. # 使用公钥加密
163. print("-------------------------加密参数------------------------")
164. C = encrypt(m, xG, yG, xB, yB, a, p, n)
165. print("密文C=\n" + str(C), end="\n\n")
166. # 使用私钥解密
167. print("------------------------解密参数------------------------")
168. decrypted\_m = decrypt(C, a, b, p, dB, len(str(hex(xG)[2:])))
169. if decrypted\_m == m:
170. print("恢复的消息与原消息一致")
171. else:
172. print("恢复的消息与原消息不一致")

四、数据分析(包括算法测试数据的分析，运行结果截图等等)

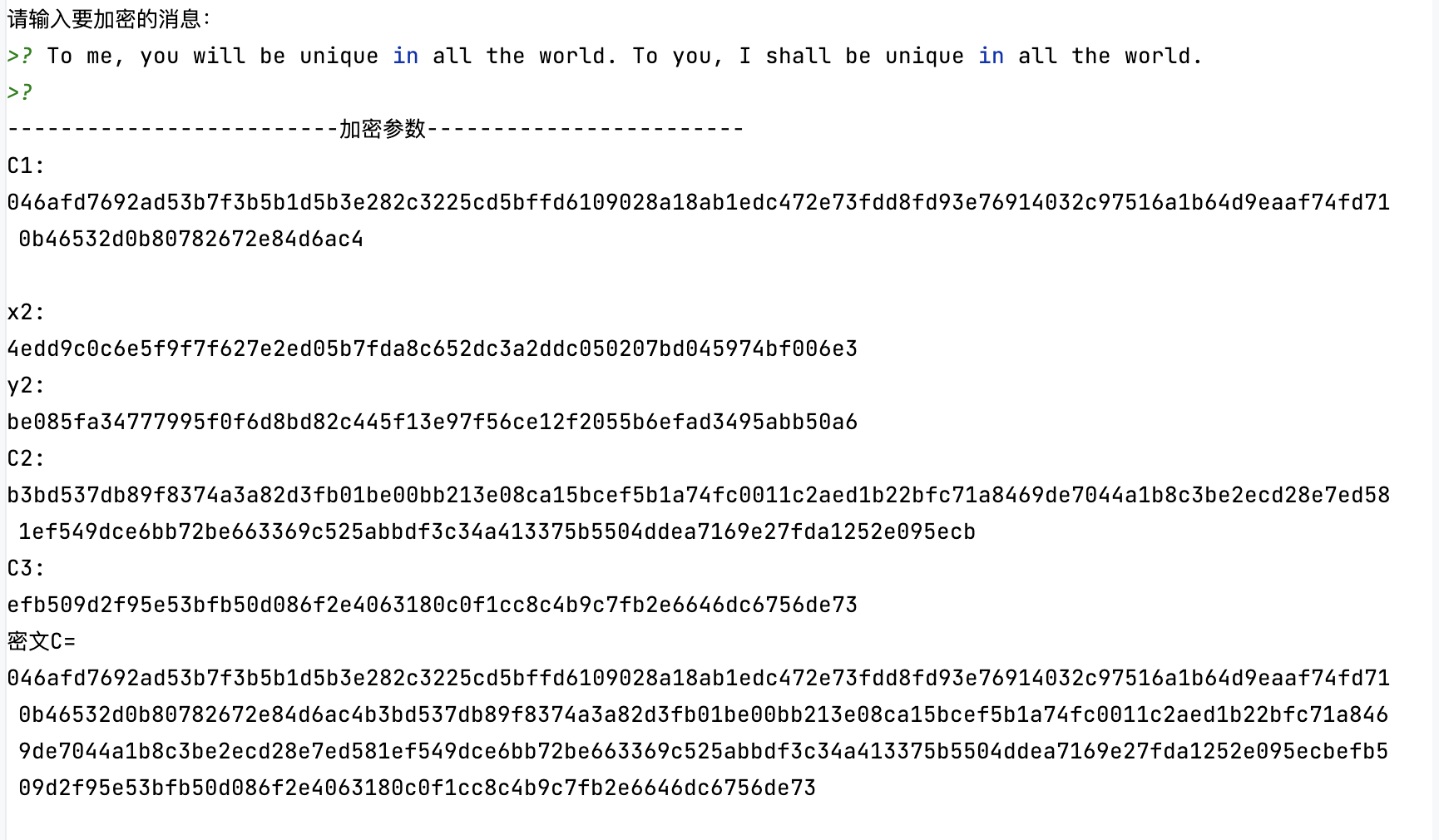
1. 算法测试数据的分析

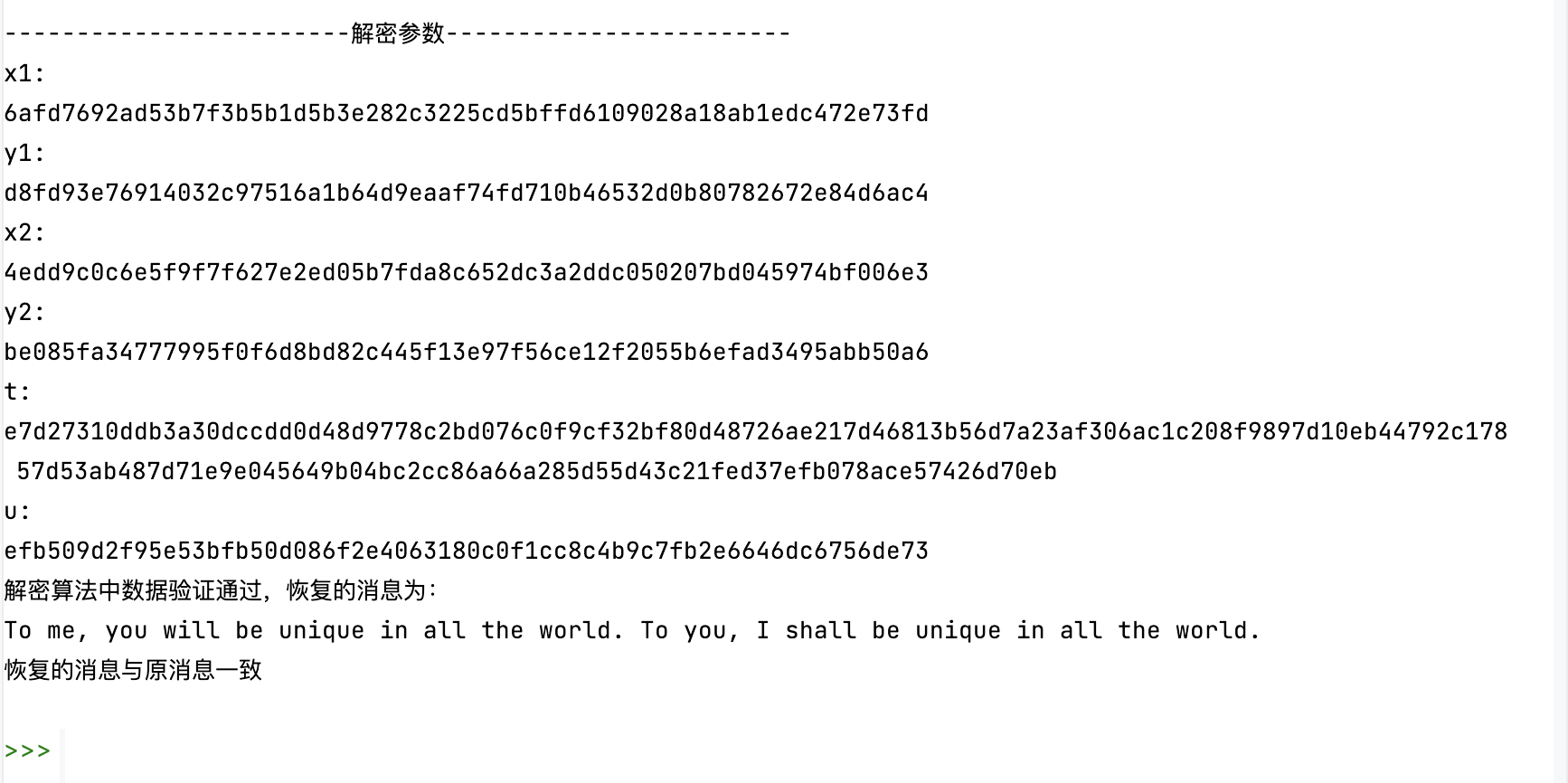
使用国家密码局所给示例中的消息进行加、解密，若正常运行则应不会报错且能够正确解密。

1. 运行结果截图

**1.txt**







**2.txt**







由图可知，程序能够正确加、解密，并能恢复消息内容。

五、思考与总结

1. 实验过程中遇到了什么问题，如何解决的？

在编写程序时，要时刻清楚此时使用的变量的数据类型，如十进制整数、二进制字符串、十六进制字符串、字符串等，若不能对使用的变量类型做到心中有数，会在编写程序时出现混乱，最终使得尽管程序编写逻辑正确，但由于数据类型不匹配，而不能正常运行。本次实验中为减少混乱的产生，对于传递的数据类型均使用十六进制字符串，必要时将其转换为十进制整数、二进制等，从而使得混乱有所减少。

在实现算法前，需要对算法的原理有所理解，不能仅根据算法步骤生搬硬套，否则在编写时，会因为对算法不理解而对算法缺少宏观把握，从而使得程序中出现许多不应出现的错误。

由于要满足1字节8位的编码规则，所以在不同数据类型相互转换时，需要确定目标数据的长度，当转换后的数据位数未达到要求位数时，填0补齐，否则在解码时会发生错误。

1. 通过该实验有何收获和感想？

通过本次实验，我对SM2国密算法有了进一步的理解，理解算法的原理及其具体加、解密算法过程，对所需辅助函数也有了进一步的了解。并且通过本次实验，对课程中所学椭圆曲线公钥密码算法有了更具体的理解，对知识有了更深入地认识。