**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（四）**

**SM2椭圆曲线公钥密码算法**

**班级：2018039**

**姓名：阿西伍合**

**学号：20179100083**

**日期：2022年12月7日**

一、实验目的（包括实验环境、实现目标等等）

实验环境

Windows11 , Anaconda python 3.8

实验目标：

1.通过算法编程，熟悉大整数的基本函数操作

2.通过编程实现SM2加密算法，理解椭圆曲线加密算法原理

二、方案设计

（包括背景、原理、必要的公式、图表、算法步骤等等）

实验背景

SM2是国家密码管理局于2010年12月17日发布的椭圆曲线公钥密码算法。公钥密码算法属于非对称加密算法，常见的非对称加密算法还有：RSA、Elgamal、背包算法、Rabin、D-H、ECC（椭圆曲线加密算法）。

非对称加密算法需要两个密钥：公开密钥（publickey:简称公钥）和私有密钥（privatekey:简称私钥）。公钥与私钥是一对，如果用公钥对数据进行加密，只能用对应的私钥才能解密；反之，用私钥对数据进行加密，只能用对应的公钥才能解密。因为具备这种独特的加解密特性，非对称加密算法不仅可实现数据的加密传输，还能对数据进行签名和验签。

实验原理

椭圆曲线

Weierstrass方程:



其中  是满足某些简单条件的域上元素。

椭圆曲线有一个特殊的点，记为，它并不在椭圆曲线上，此点称为无穷远点。

一条椭圆曲线是由全体解再加上一个无穷远点构成的集合



由Weierstrass方程给出的定义在域上的两个椭圆曲线和



被称为在域上是同构的，若存在 且 ，使得变量变换



把方程变成方程 。  式的变换称为变量的相容性变换。

设椭圆曲线的一般Weierstrass方程为



设  是曲线  上的点，则



设  是  上的两个点，且  ，则  可 由下式给出



其中



素域下的Weierstrass方程

椭圆曲线  上的点按照下面的加法运算规则，构成一个交换群：

a) ;

b) ;

c)  的逆元素  ；

d) 两个非互逆的不同点相加的规则：

设  ，

且  ，设  ，则  其中  。

e) 倍点规则: 设  ，且  ， 设  ，则  其中  。

SM2系统参数

SM2 算法使用固定的 域，系统参数如下:

-  域的特征,是m比特长度的素数；

-  中的两个元素a和b，它们定义曲线的方程:  ，a,b满足。

- 基点 (O为无穷远点)。

- 基点  的阶 是比特长度的素数。

说明:

1.  和  均为  比特长度的大整数，也可以看作 字节长度的字符串。

2. G可以看作一个有序整数对，也可以看作一个 字节长度的字符串。

3. 无穷远点O是一个理想点，不能用有序整数对  即仿射坐标表示。

推荐使用素数域256位椭圆曲线。

椭圆曲线方程：

本报告参数设置

素数p：

8542D69E 4C044F18 E8B92435 BF6FF7DE 45728391 5C45517D 722EDB8B 08F1DFC3

系数a：

787968B4 FA32C3FD 2417842E 73BBFEFF 2F3C848B 6831D7E0 EC65228B 3937E498

系数b：

63E4C6D3 B23B0C84 9CF84241 484BFE48 F61D59A5 B16BA06E 6E12D1DA 27C5249A

基点G=(xG,yG)，其阶记为n。坐标xG：

421DEBD6 1B62EAB6 746434EB C3CC315E 32220B3B ADD50BDC 4C4E6C14 7FEDD43D

坐标yG：

0680512B CBB42C07 D47349D2 153B70C4 E5D7FDFC BFA36EA1 A85841B9 E46E09A2

算法步骤

**加密算法**

输入：长度为klen的比特串M, 公钥PB

输出：SM2结构密文比特串C

算法：

1) 产生随机数k, k的值从1到n-1；

2) 计算椭圆曲线点C1=[k]G=(x1,y1), 将C1转换成比特串；

3) 验证公钥PB, 计算S=[h] PB，如果S是无穷远点，出错退出；

4) 计算(x2,y2)=[k] PB

5) 计算t=KDF(x2||y2, klen), KDF是密钥派生函数，如果t是全0比特串，返回第1）步。

6) 计算C2=M⊕t

7) 计算C3=Hash(x2||M||y2)

8) 输出密文C=C1||C3||C2, C1和C3的长度是固定的，C1是64字节，C3是32字节，很方便C从中提取C1，C3和C2。

注：通过密钥派生函数计算，才能进行第6)步的按位异或计算。

**解密算法**

输入：SM2结构密文比特串C,私钥dB

输出：明文M’

算法：

1) 从密文比特串C=C1||C3||C2中取出C1， 将C1转换成椭圆曲线上的点；

2) 验证C1, 计算S=[h] C1，如果S是无穷远点，出错退出；

4) 计算(x2,y2)=[dB] C1

5) 计算t=KDF(x2||y2, klen), KDF是密钥派生函数，如果t是全0比特串，出错退出。

6) 从C=C1||C3||C2中取出C2，计算M’= C2⊕t。

7) 计算u=Hash(x2||M’||y2)，比较u是否与C3相等，不相等则退出。

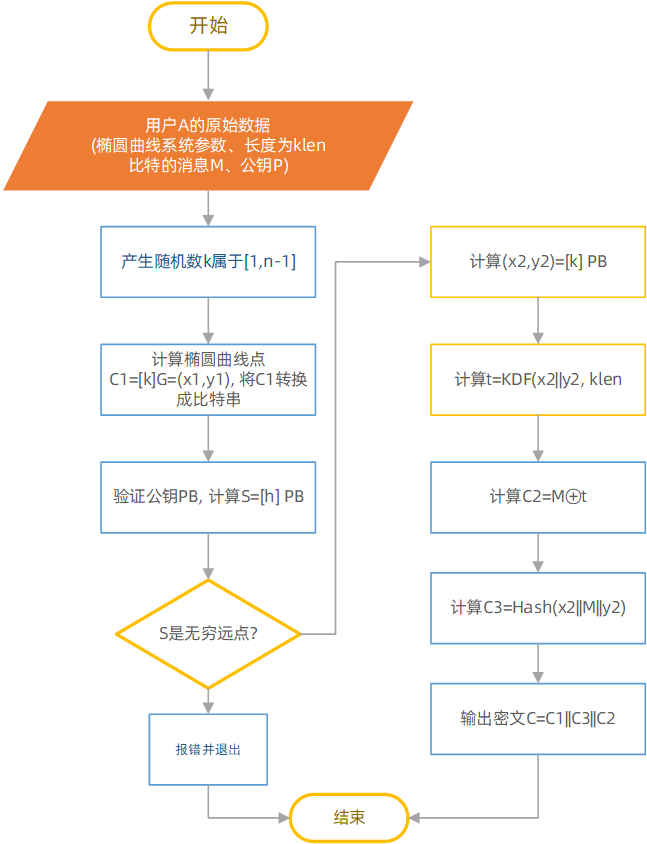
8) 输出明文M’。

三、方案实现

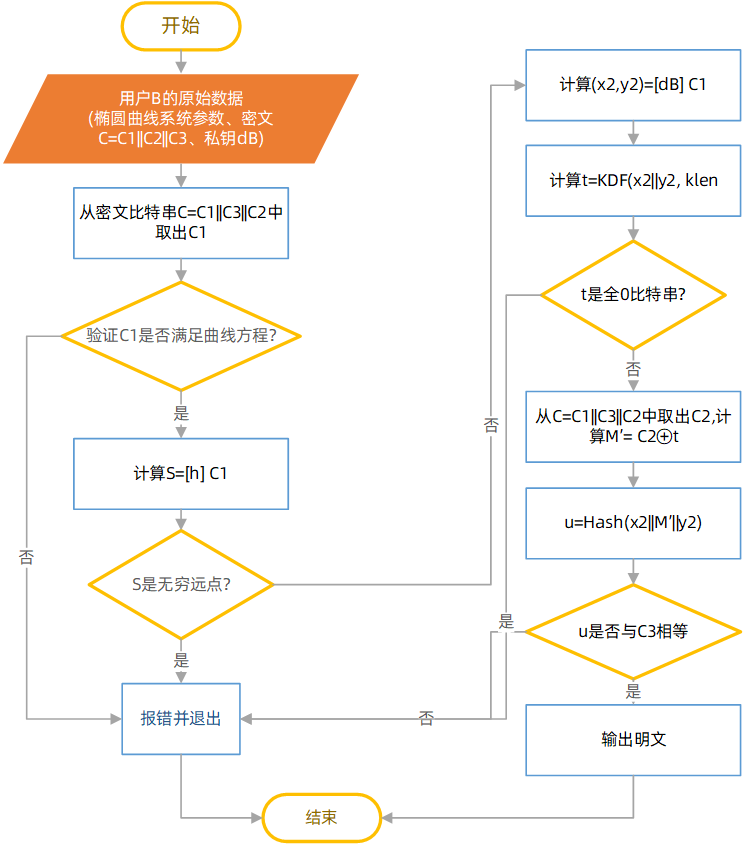
（包括算法流程图、主要函数的介绍、算法实现的主要代码等等）

算法流程图

加密

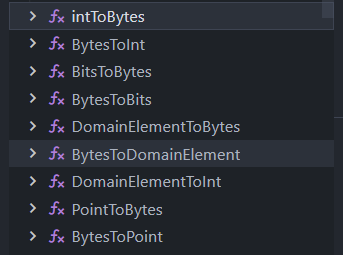


解密

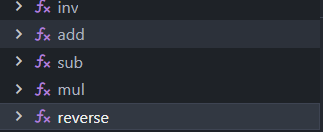


主要函数介绍

定义数据转化



在Fp素数域上的四则运算



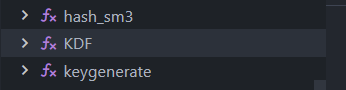
求一般式方程的参数



在Fp素数域上定义两个点的椭圆曲线点加运算和K倍点问题



我们调用了sm3库中的hash函数，实现算法中的杂凑映射功能，还实现秘钥派生函数和秘钥对生成函数



最后是加密解密函数

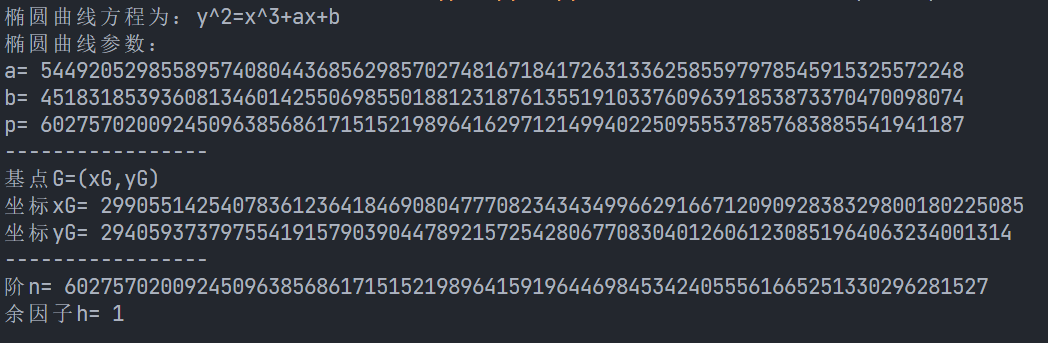


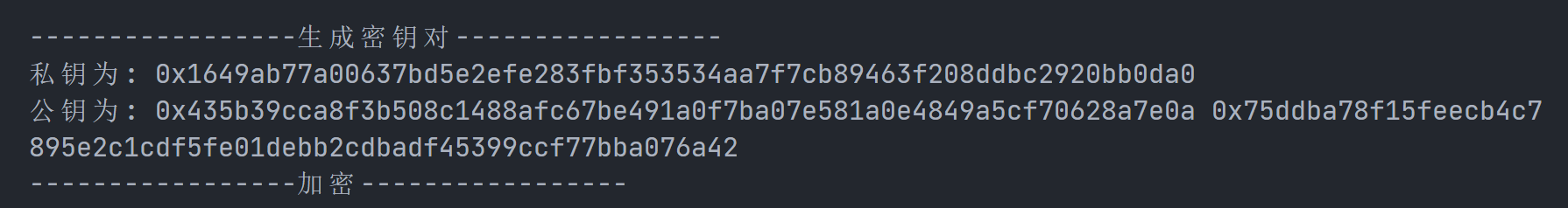
算法代码实现

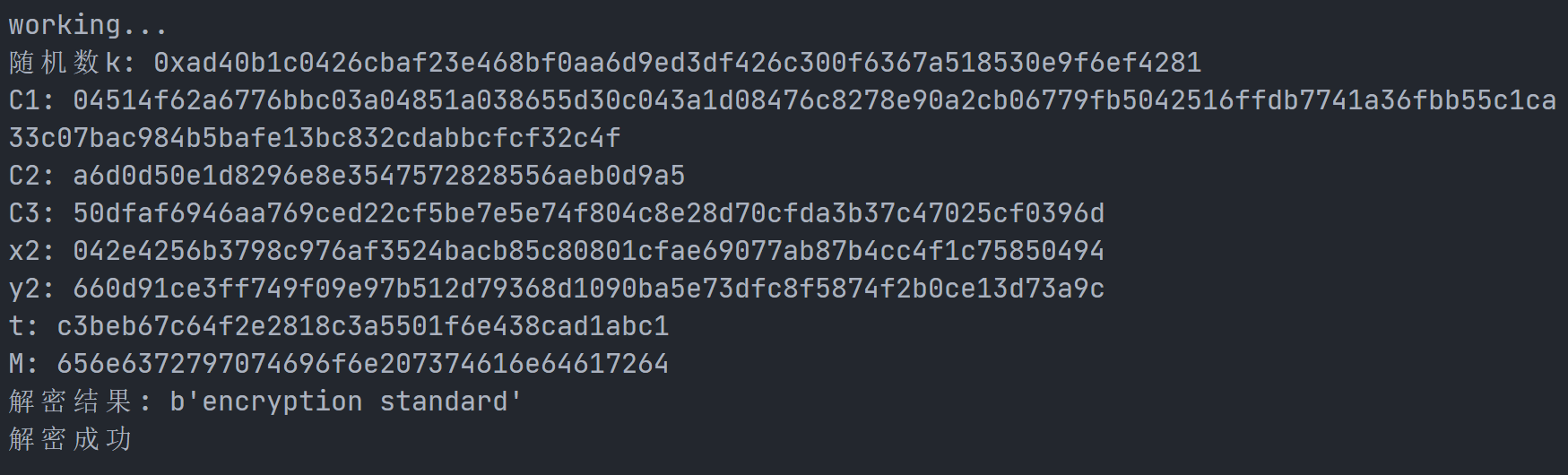
1. import hashlib
2. import math
3. import random
4. from gmpy2 import \*
5. *# 无穷远点定义*
6. INFINITY\_POINT = (None, None)
7. *# 椭圆曲线初始化*
8. compress = 0
9. ecc\_a, ecc\_b, ecc\_p, ecc\_G, ecc\_n, ecc\_h, ecc\_v = 0, 0, 0, (0, 0), 0, 0, 0
10. '''
11. :Fp-256 参考来自文档page(90/93)
12. :素数p
13. :8542D69E 4C044F18 E8B92435 BF6FF7DE 45728391 5C45517D 722EDB8B 08F1DFC3
14. :系数a
15. :787968B4 FA32C3FD 2417842E 73BBFEFF 2F3C848B 6831D7E0 EC65228B 3937E498
16. :系数b
17. :63E4C6D3 B23B0C84 9CF84241 484BFE48 F61D59A5 B16BA06E 6E12D1DA 27C5249A
18. :基点G=(xG,yG)，其阶记为n。
19. :坐标xG
20. :421DEBD6 1B62EAB6 746434EB C3CC315E 32220B3B ADD50BDC 4C4E6C14 7FEDD43D
21. :坐标yG
22. :0680512B CBB42C07 D47349D2 153B70C4 E5D7FDFC BFA36EA1 A85841B9 E46E09A2
23. :阶n
24. :8542D69E 4C044F18 E8B92435 BF6FF7DD 29772063 0485628D 5AE74EE7 C32E79B7
25. '''
26. def get\_curve(compress: int = 0):
27. *# 椭圆曲线参数*
28. ecc\_a = 0x787968B4FA32C3FD2417842E73BBFEFF2F3C848B6831D7E0EC65228B3937E498
29. ecc\_b = 0x63E4C6D3B23B0C849CF84241484BFE48F61D59A5B16BA06E6E12D1DA27C5249A
30. *# 域的规模q=p,p是大于3的素数*
31. ecc\_p = 0x8542D69E4C044F18E8B92435BF6FF7DE457283915C45517D722EDB8B08F1DFC3
32. print("椭圆曲线方程为：y^2=x^3+ax+b")
33. print("椭圆曲线参数：")
34. print("a=", ecc\_a)
35. print("b=", ecc\_b)
36. print("p=", ecc\_p)
37. print("-----------------")
38. *# 基点G*
39. xG = 0x421DEBD61B62EAB6746434EBC3CC315E32220B3BADD50BDC4C4E6C147FEDD43D
40. yG = 0x0680512BCBB42C07D47349D2153B70C4E5D7FDFCBFA36EA1A85841B9E46E09A2
41. *# 基点G=(xG,yG) 且G!=O*
42. ecc\_G = (xG, yG)
43. print("基点G=(xG,yG)")
44. print("坐标xG=", xG)
45. print("坐标yG=", yG)
46. print("-----------------")
47. *# 阶n*
48. ecc\_n = 0x8542D69E4C044F18E8B92435BF6FF7DD297720630485628D5AE74EE7C32E79B7
49. *# 选项余因子*
50. ecc\_h = 1
51. print("阶n=", ecc\_n)
52. print("余因子h=", ecc\_h)
53. *# 椭圆曲线位数*
54. ecc\_v = 256
55. print()
56. return ecc\_a, ecc\_b, ecc\_p, ecc\_G, ecc\_n, ecc\_h, ecc\_v
57. *# 数据类型转换*
58. def intToBytes(x: int, k: int):
59. '''
60. 输入:非负整数x,以及字符串的目标长度k(其中k满足2^(8k)>x))
61. 输出:长度为k的字节串M
62. '''
63. M = b''
64. type(k)
65. for \_shift in range(k):
66. *# 将x的低8位取出,转化为字节,并与M拼接*
67. M = bytes([x >> 8\*\_shift & 0xFF])+M
68. return M
69. def BytesToInt(M: bytes):
70. '''
71. 输入:长度为k的字节串M
72. 输出:非负整数x
73. '''
74. x = 0
75. for \_shift in range(len(M)):
76. *# 将M的每个字节取出,转化为整数,并与x拼接*
77. x = x << 8 | M[\_shift]
78. return x
79. def BitsToBytes(s: str):
80. '''
81. 输入:长度为m的比特串s
82. 输出:长度为m/8的字节串M
83. '''
84. length = len(s)
85. k = math.ceil(length/8)
86. *# print("k=",k)*
87. s = s.zfill(k\*8)
88. *# print("s=", s)*
89. M = b''
90. for i in range(k):
91. M = bytes([int(s[-(i\*8+8):][:8], 2)])+M
92. *# M+=bytes([int(s[-(i\*8+8):][:8], 2)])*
93. *# 因为+=是将bytes([int(s[-(i\*8+8):][:8], 2)])的值赋给M，而不是将M的值赋给bytes([int(s[-(i\*8+8):][:8], 2)])*
94. *# print("M=", M)*
95. return M
96. def BytesToBits(M: bytes):
97. '''
98. 输入:长度为m的字节串M
99. 输出:长度为m\*8的比特串s
100. '''
101. s = ""
102. M = list(reversed(M))
103. for i in range(len(M)):
104. s = (bin(M[i])[2:]).zfill(8)+s
105. return s
106. def DomainElementToBytes(a, l: int = 0):
107. '''
108. 输入:域元素a
109. 输出:长度 l =⌈t/8⌉的字节串S，其中t =⌈log2q⌉
110. '''
111. q = ecc\_p
112. *# q为素数*
113. if gmpy2.is\_prime(q) and q > 2:
114. assert a >= 0 and a <= q-1
115. assert isinstance(a, int)
116. l = math.ceil(math.log2(q)/8) if not l else l
117. S = intToBytes(a, l)
118. return S
119. raise ValueError("a的取值范围为[0,q-1],q为奇素数")
120. def BytesToDomainElement(S: bytes):
121. '''
122. 输入:长度 l =⌈t/8⌉的字节串S，其中t =⌈log2q⌉
123. 输出:F(q)中域元素a
124. '''
125. q = ecc\_p
126. *# q为奇素数*
127. if gmpy2.is\_prime(q) and q > 2:
128. a = BytesToInt(S)
129. assert a >= 0 and a <= q-1
130. return a
131. raise ValueError("a的取值范围为[0,q-1],q为奇素数")
132. def DomainElementToInt(a):
133. '''
134. 输入:域元素a
135. 输出:整数x
136. '''
137. q = ecc\_p
138. *# q为奇素数*
139. if gmpy2.is\_prime(q) and q > 2:
140. assert isinstance(a, int)
141. return a
142. raise ValueError("a不为整数")
143. def PointToBytes(P: tuple, compress: int = 0):
144. '''
145. 输入:椭圆曲线上的点P(x,y)且P!=O
146. 输出:字节串S。若选用未压缩表示形式或混合表示形式，则输出字节串长度为2l+1；若选用压
147. 缩表示形式，则输出字节串长度为l+1。(l =⌈(log2 q)/8⌉。)
148. 0-未压缩 1-压缩 2-混合表示
149. '''
150. q = ecc\_p
151. assert compress == 0 or compress == 1 or compress == 2
152. l = math.ceil(math.log2(q)/8)
153. xP, yP = P
154. X1 = DomainElementToBytes(xP, l)
155. *# 这里默认compress为0*
156. if compress == 0:
157. Y1 = DomainElementToBytes(yP, l)
158. S = b'\x04'+X1+Y1
159. return S
160. raise ValueError
161. def BytesToPoint(S: bytes, compress: int = 0):
162. '''
163. 输入:定义Fq上椭圆曲线的域元素a、b，字节串S
164. 输出:椭圆曲线上的点P(x,y)且P!=O
165. '''
166. q = ecc\_p
167. assert compress == 0 or compress == 1 or compress == 2
168. assert len(S) > 1
169. l = math.ceil(math.log2(q)/8)
170. PC = S[0]
171. X1 = S[1:l+1]
172. Y1 = S[l+1:]
173. xP = BytesToInt(X1)
174. if (compress == 0):
175. assert PC == 0x04  *# 未压缩*
176. yP = BytesToInt(Y1)
177. *# 校验点是否在曲线上 yP^2=xP^3+axP+b mod p*
178. assert pow(yP, 2, q) == (pow(xP, 3) + ecc\_a \* xP + ecc\_b) % q
179. return (xP, yP)
180. *# 定义Fp上的椭圆曲线的加法运算*
181. def inv(x):
182. '''
183. 输入:域元素x
184. 输出:域元素x的逆元
185. '''
186. *# return gmpy2.invert(x, ecc\_p)*
187. return pow(x, ecc\_p-2, mod=ecc\_p)
188. def add(a, b):
189. '''
190. 输入:域元素a,b
191. 输出:域元素a+b
192. '''
193. return (a+b) % ecc\_p
194. def sub(a, b):
195. '''
196. 输入:域元素a,b
197. 输出:域元素a-b
198. '''
199. return (a-b) % ecc\_p
200. def mul(a, b):
201. '''
202. 输入:域元素a,b
203. 输出:域元素a\*b
204. '''
205. return (a\*b) % ecc\_p
206. def reverse(y):
207. return (-y) % ecc\_p
208. def arg\_in\_double(x: int, y: int):
209. a = mul(x, x)
210. a = mul(a, 3)+ecc\_a
211. b = mul(y, 2)
212. return mul(a, inv(b))
213. def arg\_in\_add(x1: int, y1: int, x2: int, y2: int):
214. a = sub(y2, y1)
215. b = sub(x2, x1)
216. return mul(a, inv(b))
217. def PointAdd(P: tuple, Q: tuple):
218. '''
219. 输入:椭圆曲线上的点P(xP,yP)和Q(xQ,yQ)
220. 输出:椭圆曲线上的点R(xR,yR)
221. '''
222. *# 无穷远点*
223. if (P == INFINITY\_POINT):
224. return Q
225. elif Q == INFINITY\_POINT:
226. return P
227. assert isinstance(P, tuple) and isinstance(Q, tuple)
228. xP, yP = P
229. xQ, yQ = Q
230. if xP == xQ:
231. *# 互逆*
232. if reverse(yP) == yQ:
233. return INFINITY\_POINT
234. else:
235. \_lambda = arg\_in\_double(xP, yP)
236. else:
237. *# lambda=(yQ-yP)/(xQ-xP)*
238. \_lambda = arg\_in\_add(xP, yP, xQ, yQ)
239. *# x3=(lambda^2-xP-xQ) mod p*
240. x3 = sub(mul(\_lambda, \_lambda), add(xP, xQ))
241. *# y3=(lambda(xP-x3)-yP) mod p*
242. y3 = sub(mul(\_lambda, sub(xP, x3)), yP)
243. return (x3, y3)
244. *# 多倍点算法*
245. def k\_times\_point(k: int, P: tuple):
246. '''
247. 输入:整数k和椭圆曲线上的点P(xP,yP)
248. 输出:椭圆曲线上的点Q(xQ,yQ)
249. '''
250. if (P == INFINITY\_POINT):
251. return INFINITY\_POINT
252. assert isinstance(P, tuple)
253. *# 二进制展开法*
254. k = list(reversed(bin(k)[2:]))
255. l = len(k)
256. Q = INFINITY\_POINT
257. *# j从l-1到0*
258. for j in range(l-1, -1, -1):
259. Q = PointAdd(Q, Q)
260. if k[j] == '1':
261. Q = PointAdd(Q, P)
262. return Q
263. def bit\_xor(x: str, y: str):
264. maxlen = max(len(x), len(y))
265. x.ljust(maxlen, '0')
266. y.ljust(maxlen, '0')
267. res = "".join([str(int(x[i]) ^ int(y[i])) for i in range(maxlen)])
268. return res
269. def hash\_sm3(data: str) -> int:
270. '''
271. 输入:字符串data
272. 输出:字符串hash
273. '''
274. data = BitsToBytes(data)  *# 字符串转字节串*
275. sm3 = hashlib.new('sm3')  *# 创建SM3对象*
276. sm3.update(data)  *# 传入数据*
277. *# hash = sm3.hexdigest()  # 获取hash值*
278. *# print(sm3.digest())*
279. return BytesToBits(sm3.digest())  *# 字节串转字符串*
280. *# 秘钥派生函数*
281. def KDF(Z: str, klen: int):
282. '''
283. 输入:比特串Z，整数klen(表示要获得的密钥数据的比特长度，要求该值小于(2^32-1)v)
284. 输出:长度为klen的密钥数据比特串K
285. '''
286. *# 计数器32bit*
287. ct = 0x00000001
288. v = ecc\_v
289. Ha = {}
290. for i in range(1, math.ceil(klen/v)+1):
291. *# Hai=Hv(Z||ct)*
292. Ha[i] = hash\_sm3(Z+bin(ct)[2:].zfill(32))
293. ct += 1
294. if klen % v != 0:
295. *# 将Hv(Z||ct)的前前klen-(v\*math.ceil(klen/v))比特作为密钥数据的一部分*
296. Ha[math.ceil(klen/v)] = Ha[math.ceil(klen/v)
297. ][:klen-v\*math.ceil(klen/v)]
298. K = ''
299. *# 将Hai的比特串连接起来*
300. for i in range(1, math.ceil(klen/v)+1):
301. K += Ha[i]
302. return K
303. def keygenerate():
304. '''
305. 输入:无
306. 输出(d,P)其中d为私钥,P为公钥。
307. '''
308. *# 随机生成一个大于1小于n-2的整数d[1,n-2]*
309. d = random.randint(1, ecc\_n-2)
310. d = 0x1649AB77A00637BD5E2EFE283FBF353534AA7F7CB89463F208DDBC2920BB0DA0
311. P = k\_times\_point(d, ecc\_G)
312. print('私钥为:', hex(d))
313. print('公钥为:', hex(P[0]), hex(P[1]))
314. return d, P
315. def encrypt(pub: tuple, M: bytes):
316. '''
317. 输入:明文比特串M,长度为klen
318. 输出:密文C
319. '''
320. M = BytesToBits(M)
321. klen = len(M)
322. PB = pub
323. while True:
324. print("working...")
325. *# 随机生成一个大于1小于n-1的整数k*
326. k = random.randint(1, ecc\_n)
327. C1 = k\_times\_point(k, ecc\_G)
328. *#(47090616174956914622899996468915772943882005751189840835333217072271144997738, 54168491087664091928749728426261095331246089435030531539076541181117877462243)*
329. C1 = BytesToBits(PointToBytes(C1))
330. *# 
331. *# S=[h]PB 若S是无穷远点，则报错*
332. assert k\_times\_point(ecc\_h, PB) != INFINITY\_POINT
333. *# 计算[k]PB=(x2,y2)*
334. x2, y2 = k\_times\_point(k, PB)
335. x2 = BytesToBits(DomainElementToBytes(x2))
336. *# 0110001010101000111111001100000011011110011111001001111011101011000001100001101011010000000000000011111000011000100110000011001011111010001110000010000000000110010010101111111010111011100010001000001110101001110101111100010000110101101000011001100000010100*
337. y2 = BytesToBits(DomainElementToBytes(y2))
338. *# 0001001011111111100011000011111110010010000110010001101110010011111010000111001010001001100111011000001010110011111001010110011001011100010000011101001001111110010100101010100101011110011110001011011000100000011010101111011011110001000001111010010011010100*
339. *# 计算t=KDF(x2||y2,klen)*
340. t = KDF(x2+y2, klen)
341. *# print('t:',t)*
342. *# 若t全为0比特串则返回第一步*
343. if t.find("1") != -1:
344. break
345. *# 计算C2=M^t*
346. C2 = bit\_xor(M, t)
347. *# C3=H(x2||M||y2)*
348. C3 = hash\_sm3(x2+M+y2)
349. *# 返回密文C=C1||C2||C3*
350. C = C1+C2+C3
351. print("随机数k:", hex(k))
352. print("C1:", BitsToBytes(C1).hex())
353. print("C2:", BitsToBytes(C2).hex())
354. print("C3:", BitsToBytes(C3).hex())
355. print("x2:", BitsToBytes(x2).hex())
356. print("y2:", BitsToBytes(y2).hex())
357. print("t:", BitsToBytes(t).hex())
358. print("M:", BitsToBytes(M).hex())
359. return BitsToBytes(C)
360. def decrypt(pri, C: bytes):
361. C = BytesToBits(C)
362. dB = pri
363. p = ecc\_p
364. if compress == 0:
365. C1\_len = 8\*(2\*math.ceil(math.log2(p)/8)+1)
366. C1 = C[:C1\_len]
367. C3\_len = ecc\_v
368. C2\_len = len(C)-C1\_len-C3\_len
369. *# 验证C1是否在椭圆曲线上*
370. C1 = BytesToPoint(BitsToBytes(C1))
371. *# 计算椭圆曲线点S=[h]C1*
372. S = k\_times\_point(ecc\_h, C1)
373. assert S != INFINITY\_POINT
374. *# 计算[dB]C1=(x2,y2)*
375. x2, y2 = k\_times\_point(dB, C1)
376. x2 = BytesToBits(DomainElementToBytes(x2))
377. y2 = BytesToBits(DomainElementToBytes(y2))
378. *# 计算t=KDF(x2||y2,klen)*
379. t = KDF(x2+y2, C2\_len)
380. *# 若t全为0比特串则报错*
381. assert t.find("1") != -1
382. *# 计算M=C2^t*
383. C2 = C[C1\_len:C1\_len+C2\_len]
384. M = bit\_xor(C2, t)
385. *# 验证u=H(x2||M||y2)若u!=C3则报错*
386. u = hash\_sm3(x2+M+y2)
387. C3 = C[C1\_len+C2\_len:]
388. assert u == C3
389. return BitsToBytes(M)
390. def test():
391. hash\_sm3("11111111101111111")
392. def main():
393. *# 初始化椭圆曲线参数*
394. global ecc\_a, ecc\_b, ecc\_p, ecc\_G, ecc\_n, ecc\_h, ecc\_v
395. ecc\_a, ecc\_b, ecc\_p, ecc\_G, ecc\_n, ecc\_h, ecc\_v = get\_curve()
396. *# 生成密钥对*
397. print("-----------------生成密钥对-----------------")
398. d, P = keygenerate()
399. *# 明文*
400. M = b'encryption standard'
401. *# 加密*
402. print("-----------------加密-----------------")
403. C = encrypt(P, M)
404. *# 解密*
405. Message\_decrypted = decrypt(d, C)
406. print("解密结果:", Message\_decrypted)
407. if Message\_decrypted == M:
408. print("解密成功")
409. if \_\_name\_\_ == "\_\_main\_\_":
410. main()

四、数据分析(包括算法测试数据的分析，运行结果截图等等)

运行结果







五、思考与总结

1. 实验过程中遇到了什么问题，如何解决的？
2. 转换成字节串注意比特填充，否则在进制转换时会造成位丢失或位增长

严格把握格式

2）加解密中，加密时椭圆曲线点C1转换方式必须和解密时椭圆曲线点C1转换方式一致，否则无法解出C1。

1. 通过该实验有何收获和感想？

SM2基于椭圆曲线上点群离散对数难题，同样的计算复杂度，SM2的计算量相对RSA较小，处理速度快，效率高。而RSA现在在不断的增加模长，目前都用到了2048位。虽然它的安全性更高，破解难度更大，但成本也在增加。

**实验报告提交说明**：

1. 实验报告同时提交word文档与源代码（.c或者.py）。
2. 实验报告与源代码命名规则：实验5-学号-姓名，例如：实验5-20009200400-张三。
3. 请于截止日期前在西电智课平台（学在西电）提交相关文档，逾期未提交，该部分成绩记为0分，周知。