**《密码学》课程设计实验报告**

实验序号：05　　　　　　　　　　实验项目名称：公钥密码RSA

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 学　　号 |  | 姓　　名 |  | | 专业、班 | 21信安 |
| 实验地点 | C202 | 指导教师 | 余荣威 | | 时间 | 2022.12.22 |
| 1. 实验目的及要求   实验目的：   1. 掌握公钥密码的概念和基本工作方式； 2. 掌握RSA密码、ElGamal密码和椭圆曲线密码的原理与算法； 3. 了解RSA密码、ElGamal密码和椭圆曲线密码的安全性； 4. 了解RSA密码、ElGamal密码和椭圆曲线密码的应用。   实验要求：   1. 掌握RSA密码的实现方案； 2. 掌握ElGamal密码的实现方案； 3. 掌握椭圆曲线密码的实现方案； 4. 了解公钥算法实现中的相关优化算法。   二、实验设备（环境）及要求  Windows操作系统，高级语言开发环境  三、实验内容与步骤  1. RSA密码  ①随机地选择两个大素数p和q，而且保密；  ②计算n=pq，将n公开；  ③计算φ(n)=(p-1)(q-1)，对φ(n)保密；  ④随机地选取一个正整数e，1<e<φ(n)且（e，φ(n)）=1，将e公开；  ⑤根据ed＝1 mod φ(n)，求出d，并对d保密；  ⑥加密运算：  C＝Me mod n （7-4）  ⑦解密运算：  M＝Cd mod n （7-5）  **实验（1）**令p=3,q=11,d=7,m=5,手工或编程计算密文C 。  **实验（2）**设RSA密码的 e=3,n=33,C=9, 手工或编程计算明文M 。  **实验（3）**令p=17,q=11, e=7,试计算RSA密码其余参数 。  进一步对于m=88, 计算密文C 。  2. ELGamal密码（参见教材p219）  **例：**设p=19，m=17,构造一个ELGamal密码，并用它对m加密。  **实验（4）**设p=5，m=3,构造一个ELGamal密码，并用它对m加密。  3.椭圆曲线密码（选作）  （1）GF(p)上的椭圆曲线  **实验（5）**取p=23,求出椭圆曲线 y2=x3+x+1的全部解点。（选作）  （2）椭圆曲线密码  理解并实现SM2算法加解密过程。（教材p239）  四、实验结果与数据处理  4.1 RSA密码  RSA密码已在理论课实现  4.2 ELGamal密码  ELGamal已在理论课实现，这里因为提前写了所以就留下了ELGamal的部分   1. 加密过程如下：     对应的代码实现：     1. 解密过程如下：     对应的代码实现:    （3）实验结果验证  ①对例题进行验证：  **例：**设p=19，m=17,构造一个ELGamal密码，并用它对m加密。    ②对实验（4）进行验证  实验（4）设p=5，m=3,构造一个ELGamal密码，并用它对m加密。  实验结果如下：    4.3 椭圆曲线密码（选作）  （一）GF(p)上的椭圆曲线  （1）加法定义  P=Q时    P≠Q时    对应的代码实现如下：   1. def add(x1, y1, x2, y2, a, p): 3. flag = 1  # 定义符号位（+/-） 5. # 如果 p=q  k=(3x2+a)/2y1mod p 6. **if** x1 == x2 and y1 == y2: 7. molecule = 3 \* (x1 \*\* 2) + a  # 计算分子 8. denominator = 2 \* y1    # 计算分母 10. # 若P≠Q，则k=(y2-y1)/(x2-x1) mod p 11. **else**: 12. molecule = y2 - y1 13. denominator = x2 - x1 14. **if** molecule\* denominator < 0: 15. flag = 0        # 符号0为-（负数） 16. molecule = abs(molecule) 17. denominator = abs(denominator) 19. # 将分子和分母化为最简 20. gcd\_value = get\_gcd(molecule, denominator) 21. molecule = molecule // gcd\_value 22. denominator = denominator // gcd\_value 23. #求molecule / denominator = molecule \* inverse\_denominator 24. inverse\_denominator = get\_inverse(denominator, p) 25. k = (molecule \* inverse\_denominator) 27. **if** flag == 0:                   # 斜率负数 flag==0 28. k = -k 29. k = k % p 30. # 计算x3,y3 P+Q 31. """ 32. x3≡k2-x1-x2(mod p) 33. y3≡k(x1-x3)-y1(mod p) 34. """ 35. x3 = (k \*\* 2 - x1 - x2) % p 36. y3 = (k \* (x1 - x3) - y1) % p 37. **return** x3,y3   （2）逆元    对应代码如下：     1. 倍点运算     对应代码如下：   1. def get\_ng(G\_x, G\_y, n, a, p): 2. """ 3. 计算nG 4. """ 5. temp\_x = G\_x 6. temp\_y = G\_y 7. **while** n != 1: 8. temp\_x,temp\_y = add(temp\_x,temp\_y, G\_x, G\_y, a, p) 9. n -= 1 10. **return** temp\_x,temp\_y   4）求解椭圆曲线的阶   1. def get\_rank(x0, y0, a, b, p): 2. """ 3. 获取椭圆曲线的阶: 4. 获取n\*p，每次+p，直到求解阶数np=-p 6. """ 7. #p =(x,y) -p = (x,-y) 8. x1 = x0             #-p的x坐标 9. y1 = (-1\*y0)%p      #-p的y坐标 10. tempX = x0 11. tempY = y0 12. n = 1 13. **while** True: 14. n += 1 15. # 求p+q的和，得到n\*p，直到求出阶 16. p\_x,p\_y = add(tempX, tempY, x0, y0, a, p) 17. # 如果 == -p,那么阶数+1，返回 18. **if** p\_x == x1 and p\_y == y1: 19. **return** n+1 20. tempX = p\_x 21. tempY = p\_y 23. **return** n   （5）加解密过程     1. # 获取椭圆曲线的阶 2. n = get\_rank(G\_x, G\_y, a, b, p) 4. # userA生成私钥，小key 5. key = **int**(input("userA：请输入私钥key（<{}）：".format(n))) 7. # userA生成公钥，大Q=key G 8. Q\_x,Q\_y = get\_ng(G\_x, G\_y, key, a, p)     对应代码如下：  userB阶段      # userB拿到userA的公钥KEY，Ep(a,b)阶n，加密需要加密的明文数据      # 加密准备      k = random.randint(1, n)  # 生成1到阶n之间的随机整数      print("随机产生的整数K，用于加密数据：",k)      #k \* G      k\_G\_x,k\_G\_y = get\_ng(G\_x, G\_y, k, a, p)                         # kG      #k \* Q      k\_Q\_x,k\_Q\_y = get\_ng(Q\_x, Q\_y, k, a, p)                     # kQ      # 加密      plain\_text = input("userB：请输入需要加密的字符串:")      plain\_text = plain\_text.strip()      c = []      print("密文为：",end="")      for char in plain\_text:          #获取每个字符的ASCII码值          intchar = ord(char)          #将ASCII码值乘以密钥k\_Q\_x的得到密文 ASCII \* r \* KEY          cipher\_text = intchar\*k\_Q\_x          c.append([k\_G\_x, k\_G\_y, cipher\_text])          print("({},{}),{}".format(k\_G\_x, k\_G\_y, cipher\_text),end="-")    对应代码如下：  # userA阶段      # 拿到userB加密的数据进行解密      # 知道 k\_G\_x,k\_G\_y，key情况下，求解k\_Q\_x,k\_Q\_y是容易的，然后plain\_text = cipher\_text/k\_Q\_x      print("\nuserA解密得到明文：",end="")      for charArr in c:          #key \* kG = k \* (key \* G) = k \* Q          decrypto\_text\_x,decrypto\_text\_y = get\_ng(charArr[0], charArr[1], key, a, p)          #m \* kQ / kQ          print(chr(charArr[2]//decrypto\_text\_x),end="")  （4）验证实验（5）  实验（5）取p=23,求出椭圆曲线 y2=x3+x+1的全部解点。  a=1,b=1,p=23      （二）椭圆曲线密码  理解并实现SM2算法加解密过程。（教材p239）  （1）加密过程  加密过程的流程框图如下：    对应的加密算法步骤如下：    print("Step1：产生随机数k∈[1,n-1]")      k = random.randint(1, n-1)      k\_hex = hex(k)[2:]          # k\_hex 是k的十六进制串形式    print("\nStep2:计算椭圆曲线点C1=[k]G=(x1,y1)，将C1的数据类型转换为比特串")      C1 = mult\_point(G, k, p, a)      print("椭圆曲线点C1=[k]G=(x1,y1)的坐标是:", tuple(map(hex, C1)))      C1\_bits = point\_to\_bits(C1)      print("椭圆曲线点C1=[k]G=(x1,y1)的坐标的比特串形式是:", C1\_bits)    print("\nStep3：计算椭圆曲线点S = [h]PB")      S = mult\_point(PB, h, p, a)      if S == 0:          raise Exception("计算得到的S是无穷远点")      print("椭圆曲线点S = [h]PB的坐标是:", tuple(map(hex, S)))    print("\nStep4：计算椭圆曲线点[k]PB=(x2,y2)，将坐标x2、y2 的数据类型转换为比特串")      x2, y2 = mult\_point(PB, k, p, a)      print("椭圆曲线点[k]PB=(x2,y2)的坐标是:", tuple(map(hex, (x2, y2))))      x2\_bits = fielde\_to\_bits(x2)      print("x2的比特串形式是：", x2\_bits)      y2\_bits = fielde\_to\_bits(y2)      print("y2的比特串形式是：", y2\_bits)    print("\nStep5：计算t=KDF(x2 ∥ y2, klen)，若t为全0比特串，则返回Step1")      M\_hex = bytes\_to\_hex(M\_bytes)      klen = 4 \* len(M\_hex)      print("明文消息的比特串长度klen是：", klen)      t = KDF(x2\_bits + y2\_bits, klen)      print("通过KDF算法计算得到的t=KDF(x2 ∥ y2, klen)是：", t)      if eval('0b' + t) == 0:          raise Exception("KDF返回了全零串，请检查KDF算法！")      t\_hex = bits\_to\_hex(t)      print("t的十六进制表示形式是：", t\_hex)  其中使用到了KDF函数    对应代码实现如下：  def KDF(Z, klen):      v = 256           # 密码杂凑函数采用SM3      if klen >= (pow(2, 32) - 1) \* v:          raise Exception("密钥派生函数KDF出错，请检查klen的大小！")      ct = 0x00000001      if klen % v == 0:          l = klen // v      else:          l = klen // v + 1      Ha = []      for i in range(l):         # i从0到 klen/v-1（向上取整）,共l个元素          s = Z + int\_to\_bits(ct).rjust(32, '0')         # s存储 Z || ct 的比特串形式 # 注意，ct要填充为32位          s\_bytes = bits\_to\_bytes(s)          # s\_bytes存储字节串形式          s\_list = [i for i in s\_bytes]          hash\_hex = sm3.sm3\_hash(s\_list)          hash\_bin = hex\_to\_bits(hash\_hex)          Ha.append(hash\_bin)          ct += 1      if klen % v != 0:          Ha[-1] = Ha[-1][:klen - v\*(klen//v)]      k = ''.join(Ha)      return k  KDF在执行过程中，首先判断派生密钥长度klen是否合法，如果大于等于((2^32)-1) \* 256，则抛出异常。接着，设置初始计数器ct为1，计算派生密钥块的个数l，并初始化一个空列表Ha用于存储每个派生密钥块的中间结果。通过循环迭代，对每个派生密钥块计算并存储其对应的SM3哈希值，其中输入为输入密钥Z和32位的计数器ct。最后，根据派生密钥长度是否整数倍于256进行处理，得到最终派生密钥k，并将其作为输出返回。    print("\nStep6：计算计算C2 = M ⊕ t")      C2 = eval('0x' + M\_hex + '^' + '0b' + t)      print("计算的C2是：", hex(C2)[2:])    print("\nStep7：计算C3 = Hash(x2 ∥ M ∥ y2)")      x2\_bytes = bits\_to\_bytes(x2\_bits)      y2\_bytes = bits\_to\_bytes(y2\_bits)      hash\_list = [i for i in x2\_bytes + M\_bytes + y2\_bytes]      C3 = sm3.sm3\_hash(hash\_list)     print("\nStep8：输出密文C = C1 ∥ C2 ∥ C3")      C1\_hex = bits\_to\_hex(C1\_bits)      C2\_hex = hex(C2)[2:]      C3\_hex = C3      C\_hex = C1\_hex + C2\_hex + C3\_hex      print("加密得到的密文是：", C\_hex)  （2）解密过程  解密计算的流程框图如下：      print("Step1：从C中取出比特串C1，将C1的数据类型转换为椭圆曲线上的点，验证C1是否满足椭圆曲线方程，若不满足则报错并退出；")      l = ceil(log(p, 2)/8)         # l是一个域元素（比如一个点的横坐标）转换为字节串后的字节长度.则未压缩的形式下密文第一部分C1长度为2l+1      bytes\_l1 = 2\*l+1      print("计算得到的C1的字节串长度是：", bytes\_l1)      hex\_l1 = bytes\_l1 \* 2            # hex\_l1是密文第一部分C1的十六进制串的长度      C\_bytes = hex\_to\_bytes(C)      print("将十六进制密文串转换为字节串是：", C\_bytes)      C1\_bytes = C\_bytes[0:2\*l+1]      print("从密文字节串中取出的C1的字节串是：", C1\_bytes)      C1 = bytes\_to\_point(C1\_bytes)      print("将C1字节串转换为椭圆曲线上的点是：", C1)      if not on\_curve(args, C1):          # 检验C1是否在椭圆曲线上          raise Exception("在解密算法Step1中，取得的C1不在椭圆曲线上")      x1, y1 = C1[0], C1[1]      x1\_hex, y1\_hex = fielde\_to\_hex(x1), fielde\_to\_hex(y1)      print("C1坐标用的十六进串形式表示是：", (x1\_hex, y1\_hex))    print("\nStep2：计算椭圆曲线点S=hC1，若S是无穷远点，则报错并退出；")      S = mult\_point(C1, h, p, a)      print("计算得到的S是：", S)      if S == 0:          raise Exception("在解密算法Step2中，S是无穷远点")      xS, yS = S[0], S[1]      xS\_hex, yS\_hex = fielde\_to\_hex(xS), fielde\_to\_hex(yS)      print("S的坐标用十六进制串形式表示是：", (xS\_hex, yS\_hex))    print("\nStep3：计算dC1=(x2,y2)，将坐标x2、y2的数据类型转换为比特串；")      temp = mult\_point(C1, dB, p, a)      x2, y2 = temp[0], temp[1]      x2\_hex, y2\_hex = fielde\_to\_hex(x2), fielde\_to\_hex(y2)      print("解密得到的dC1=(x2,y2)的十六进制串形式是：", (x2\_hex, y2\_hex))    print("\nStep4:计算t=KDF(x2 ∥ y2, klen)，若t为全0比特串，则报错并退出；")      hex\_l3 = 64           # hex\_l3是密文第三部分C3的十六进制串的长度。C3是通过SM3得到的hash值，是64位十六进制串。      hex\_l2 = len(C) - hex\_l1 - hex\_l3           # hex\_l2是密文第二部分C2的十六进制串的长度。      klen = hex\_l2 \* 4           # klen是密文C2中比特串的长度      print("计算的C2的比特串长度klen是：", klen)      x2\_bits, y2\_bits = hex\_to\_bits(x2\_hex), hex\_to\_bits(y2\_hex)      t = KDF(x2\_bits + y2\_bits, klen)      print("计算的t=KDF(x2 ∥ y2, klen)是：", t)      if eval('0b' + t) == 0:          raise Exception("在解密算法Step4中，得到的t是全0串")      t\_hex = bits\_to\_hex(t)      print("t的十六进制串形式是：", t\_hex)    print("\nStep5：从C中取出比特串C2，计算M′ = C2 ⊕ t；")      C2\_hex = C[hex\_l1: -hex\_l3]      print("C2的十六进制串形式是：", C2\_hex)      M1 = eval('0x' + C2\_hex + '^' + '0x' + t\_hex)           # M1是M'，M′ = C2 ⊕ t      M1\_hex = hex(M1)[2:].rjust(hex\_l2, '0')         # 注意位数要一致      print("计算的M′ = C2 ⊕ t是：", M1\_hex)    print("\nStep6：计算u = Hash(x2 ∥ M′ ∥ y2)，从C中取出比特串C3，若u != C3，则报错并退出；")      M1\_bits = hex\_to\_bits(M1\_hex)      cmp\_bits = x2\_bits + M1\_bits + y2\_bits          # cmp\_bits存储用于计算哈希值以对比C3的二进制串      cmp\_bytes = bits\_to\_bytes(cmp\_bits)      cmp\_list = [i for i in cmp\_bytes]      u = sm3.sm3\_hash(cmp\_list)          # u中存储      print("计算的u = Hash(x2 ∥ M′ ∥ y2)是：", u)      C3\_hex = C[-hex\_l3:]      print("从C中取出的C3的十六进制形式是：", C3\_hex)      if u != C3\_hex:          raise Exception("在解密算法Step6中，计算的u与C3不同")    print("\nStep7：输出明文M′")      M\_bytes = hex\_to\_bytes(M1\_hex)      M = str(M\_bytes, encoding='ascii')      print("解密出的明文是：", M)  （3）实验结果  以教材p239的例子检验SM2加解密结果    对应代码如下：   p = eval('0x' + '8542D69E 4C044F18 E8B92435 BF6FF7DE 45728391 5C45517D 722EDB8B 08F1DFC3'.replace(' ', ''))      a = eval('0x' + '787968B4 FA32C3FD 2417842E 73BBFEFF 2F3C848B 6831D7E0 EC65228B 3937E498'.replace(' ', ''))      b = eval('0x' + '63E4C6D3 B23B0C84 9CF84241 484BFE48 F61D59A5 B16BA06E 6E12D1DA 27C5249A'.replace(' ', ''))      h = 1      xG = eval('0x' + '421DEBD6 1B62EAB6 746434EB C3CC315E 32220B3B ADD50BDC 4C4E6C14 7FEDD43D'.replace(' ', ''))      yG = eval('0x' + '0680512B CBB42C07 D47349D2 153B70C4 E5D7FDFC BFA36EA1 A85841B9 E46E09A2'.replace(' ', ''))      G = (xG, yG)            # G 是基点      n = eval('0x' + '8542D69E 4C044F18 E8B92435 BF6FF7DD 29772063 0485628D 5AE74EE7 C32E79B7'.replace(' ', ''))  运行结果如下：  加密过程：      五、分析与讨论  解密过程：  解密过程：      可以看到SM2加解密成功  五、分析与讨论  5.1 SM2算法中有哪些检错的步骤？  加密过程的步骤三配备了检错的步骤  print("\nStep3：计算椭圆曲线点S = hPB")      S = mult\_point(PB, h, p, a)      if S == 0:          raise Exception("计算得到的S是无穷远点")  首先对公钥PB进行备点运算计算椭圆曲线点S，通过检查计算得到的点 S 是否为无穷远点（O）来判断计算结果是否有效。如果 S 是无穷远点，说明计算过程中可能存在问题，可能是由于输入参数错误、攻击尝试或其他异常情况导致的计算错误。在这种情况下，抛出异常是为了中断程序的执行，避免进一步使用不安全的计算结果。  SM2在解密算法中加入了更多的检错功能：  1.检查密文C1的正确性      if not on\_curve(args, C1):          # 检验C1是否在椭圆曲线上          raise Exception("在解密算法Step1中，取得的C1不在椭圆曲线上")  通过 on\_curve 函数验证 C1 是否在椭圆曲线上。这个检查确保了解密过程的第一步中使用的 C1 确实属于椭圆曲线上的一个点，以防止椭圆曲线参数被篡改或者攻击者伪造的情况。  2.进一步检查C1的正确性  print("\nStep2：计算椭圆曲线点S=hC1，若S是无穷远点，则报错并退出；")      S = mult\_point(C1, h, p, a)      print("计算得到的S是：", S)      if S == 0:          raise Exception("在解密算法Step2中，S是无穷远点")  在计算 S = h \* C1 之后，通过检查 S 是否为无穷远点来确保计算结果有效。如果 S 是无穷远点，表示解密过程中存在异常，可能是由于攻击或其他错误导致的。  3.检查t的正确性，其中包括着C2的正确性  if eval('0b' + t) == 0:          raise Exception("在解密算法Step4中，得到的t是全0串")  计算 KDF(x2 ∥ y2, klen) 后，通过检查 t 是否为全0串来确保派生密钥的有效性。如果 t 全为0，可能会导致密钥派生过程中的错误。  4．检查C3的正确性：   if u != C3\_hex:          raise Exception("在解密算法Step6中，计算的u与C3不同")  在计算 Hash(x2 ∥ M′ ∥ y2) 后，通过与 C3 比较来确保接收到的密文在传输过程中未被篡改。如果 u 与 C3 不一致，说明可能存在攻击或者传输错误。  5.2 比较传统ECC和SM2  1.传统椭圆曲线密码只利用分量作为密钥进行加密：，而没有利用分量。而SM2利用分量和经过密钥派生函数产生中间密钥，再用进行加密：  2.传统椭圆曲线密码的加密运算是乘法，运算较为复杂，效率低。而SM2加密运算是模2加，因此效率更高。  3．传统椭圆曲线使用为密文，而SM2使用 作为密文，密文数据扩张相较于前者较为严重。  4.SM2相比于ECC增加了许多检错的措施，因此提高了密码系统的数据完整性和系统可靠性，进而提高了密码系统的安全性。 | | | | | | |
|  | | | | | | |
| 六、教师评语  签名：  日期： | | | | 成绩 | | |