**《密码学》课程设计实验报告**

实验序号：06　　　　　　　　　　实验项目名称：数字签名

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 学　　号 |  | 姓　　名 |  | 专业、班 | | 21信安 |
| 实验地点 | C202 | 指导教师 | 余荣威 | 时间 | | 2022.12.29 |
| 1. 实验目的及要求   实验目的：   1. 掌握数字签名的概念； 2. 掌握基于RSA密码、ElGamal密码和椭圆曲线密码的数字签名方法； 3. 了解基于RSA密码、ElGamal密码和椭圆曲线密码的数字签名的安全性； 4. 熟悉盲签名的原理，了解盲签名的应用。   实验要求：   1. 掌握RSA数字签名的实现方案； 2. 掌握ElGamal数字签名的实现方案； 3. 掌握SM2椭圆曲线数字签名的实现方案； 4. 了解数字签名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. 编程实现RSA数字签名方案  参数准备阶段：同实验07中的RSA密码算法  签名运算：  S＝Md mod n （8-5）  验证签名运算（判断式）：  M＝Se mod n （8-6）  实验1：采用实验06中的RSA密码算法的相关参数，对于M进行签名及验证。  思考1：RSA数字签名方案的几种攻击方法  思考2：基于RSA数字签名的盲签名方案的实现  2．编程实现ELGamal数字签名方案  复习数论的一个结论。对于素数*q*，如果是*q*的原根，则有： 取模(mod *q*)后各不相同。因此如果是*q*的原根，进一步有：   1. 对于任意整数m，当且仅当 2. 对于任意整数i，j，当且仅当   同ELGamal加密方案一样，ELGamal数字签名方案的基本元素是素数*p*和，其中是*p*的原根。用户A通过如下步骤产生公钥/私钥对：   1. 生成随机整数，使得。 2. 计算。   A的私钥是；A的公钥是。  **例：**设p=19，m=14,构造一个ELGamal数字签名方案，并用它对m签名。  对于p=19，原根有{2,3,10,13,14,15}，任选其中之一作为模 19的本原元（生成元），如选择  **3.1 密钥生成**        # 大素数      p = Generate\_prime(512)      # alpha是模p的本原元      alpha = Generate\_alpha(p)      # 用户自己的解密钥      x = Generate\_private\_key(p)      # 用户公开的加密钥      y = power(alpha, x, p)  **3.1.1 大素数生成算法**  def Generate\_prime(key\_size: int) -> int:      while True:          num = random.randrange(quick\_power(2, key\_size - 1), quick\_power(2, key\_size))          if Miller\_Rabin(num):              return num  该大素数生成算法在一个无限循环中，通过调用random.randrange函数生成一个在指定范围内的随机整数num，范围限定在2的key\_size-1次方到2的key\_size次方之间。然后使用Miller-Rabin素性检测算法判断生成的随机数num是否为素数，若是素数则返回该值，否则继续循环生成新的随机数进行检测，直至生成满足条件的素数。  **3.1.2 本原元生成算法**  def Generate\_alpha(p: int) -> int:      """      生成模 p 意义下的本原元      """      while True:          alpha = random.randint(2, p - 1)          if is\_primitive\_root(alpha, p):              return alpha  Generate\_alpha(p) 用于生成模 p 下的本原元。函数内部定义了辅助函数 is\_primitive\_root(alpha, p) 用于检查给定的 α 是否是模 p 下的本原元。      def is\_primitive\_root(alpha, p):          """          检查 alpha 是否是模 p 意义下的本原元          """          for i in range(2, p - 1):              if pow(alpha, i, p) == 1:                  return False          return True  在函数内部，通过循环遍历 i（取值范围2到p-1），检查modp是否等于1，如果存在i使得这个条件成立，那么α就不是模p的本原元，反之如果对所有i都有modp≠1，那么α就是模p的本原元  **3.1.3私钥生成算法**  defGenerate\_private\_key(p:int)->int:  pri=random.randint(2,p-2)  whilegcd(pri,p)!=1:  pri=random.randint(2,p-2)  returnpri  在该函数中，首先通过random.randint生成一个在区间[2,p-2]内的随机整数pri作为私钥候选。然后进入一个循环，判断当前候选私钥pri与大素数p是否互质（最大公约数为1），如果不满足互质条件，则重新生成新的随机整数作为私钥候选，直至找到符合互质条件的私钥。最终返回满足条件的私钥值。  **3.2 签名过程**    代码如下所示  # 计算签名  def Sign(m, p, alpha, x) -> []:      k = random.randint(0, p - 2)      while gcd(k, p - 1) != 1:          k = random.randint(0, p - 2)      #验证步骤二      #k = 5      r = power(alpha, k, p)      s = (m - x \* r) \* Extended\_Eulid(k, p - 1) % (p - 1)      return r, s  代码与上述实现过程是一模一样的，在这里不过多赘述了，其中k逆通过扩展的欧几里得算法实现  # 扩展的欧几里得算法，ab=1 (mod m), 得到a在模m下的乘法逆元b  def Extended\_Eulid(a: int, m: int) -> int:      def extended\_eulid(a: int, m: int):          if a == 0:  # 边界条件              return 1, 0, m          else:              x, y, gcd = extended\_eulid(m % a, a)  # 递归              x, y = y, (x - (m // a) \* y)  # 递推关系，左端为上层              return x, y, gcd  # 返回第一层的计算结果。          # 最终返回的y值即为b在模a下的乘法逆元          # 若y为复数，则y+a为相应的正数逆元      n = extended\_eulid(a, m)      if n[1] < 0:          return n[1] + m      else:          return n[1]  **3.3验证过程**    验证过程如下：  # 签名验证  def Verify(m, p, alpha, y, r, s):      v1 = power(alpha, m, p)      v2 = (power(y, r, p) \* power(r, s, p)) % p      if v1 == v2:          return True      else:          return False  3.编程实现SM2椭圆曲线数字签名方案（选作）  思考1：椭圆曲线加密、签名的快速实现；  提示1：模参数、曲线参数的选取优化；  提示2：点加和倍点运算的快速实现；  思考2：k=15时，kP运算次数  反复平方乘31P=[11111]2P=2(2（2（2P+P）+P）+P)+P，共4次加法、4次倍加  改进编码31P=（25-1）P=2(2(2(2(2P))))-P，需要5次倍加，1次加法（减法）  四、实验结果与数据处理  编程实现ELGamal数字签名方案结果，具体的算法细节详见第三节  **4.1密钥生成**  用户随机地选择一个整数x作为自己的秘密的解密钥，1<x<p-1，计算y≡αxmodp，取y为自己的公开的加密钥。例如选择，，即私钥为16，公钥为4.  为了验证上述步骤，我手动选择了用户的解密钥x=16，本原元α=10，大素数p=19  p=19  alpha=10  x=16  y=power(alpha,x,p)  可以得到结果如下：    可以看到私钥为16，公钥为4，与上述内容一致  **4.2 签名过程**  将明文消息M（0≤M≤p-1)加密成密文的过程如下：   1. 随机地选取一个整数k，k与p-1互素且1≤k≤p-1。例如随机选择 2. 计算 3. 取（r，s）=（3,4）作为m=14的签名。   为了验证上述内容，在这里我手动选择k=5，m=14   k = 5      r = power(alpha, k, p)      s = (m - x \* r) \* Extended\_Eulid(k, p - 1) % (p - 1)  运行    可以看到(r,s)=(3,4)作为14的前面，正确  **4.3验证过程**  对签名（r，s）验证的过程如下：   1. 计算 2. 计算   由于，所以签名是合法的。    可以看到结果是符合验证过程的  **4.4 变形算法的签名验证**  任意选作教材p254表8-1中的数字签名的变形算法，对于M进行签名及验证。    上表中4就是我们的ELGamal签名算法，在这里我选择2作为我们的签名算法    因此签名算法修改如下：  # 计算签名  def Sign(m, p, alpha, x) -> []:      k = random.randint(0, p - 2)      while gcd(k, p - 1) != 1:          k = random.randint(0, p - 2)      #验证步骤二      k = 5      r = power(alpha, k, p)      #s的计算部分进行修改      s = (m \* x - r) \* Extended\_Eulid(k, p - 1) % (p - 1)      return r, s  验签部分修改如下：  # 签名验证  def Verify(m, p, alpha, y, r, s):      #v1,v2部分进行修改      v1 = power(y, m, p)      v2 = (power(alpha, r, p) \* power(r, s, p)) % p      print(f"v1: {v1}")      print(f"v2: {v2}")      if v1 == v2:          return True      else:          return False  依然采用上面的参数进行验证    可以看到我们的签名验证有效  五、分析与讨论  5.1 ELGamal 安全性分析  ELGamal签名面临存在性的伪造攻击，攻击方案如下所示：    解决方案：不直接对m进行签名，而是对Hash(m)进行签名  5.2 ELGamal与RSA数字签名的比较  RSA签名算法： RSA签名算法是一种基于RSA公钥密码体系的数字签名算法，其基本思想是利用RSA算法的可逆性和不可逆性来实现数字签名的功能。签名者使用自己的私钥对消息进行签名，验证者使用签名者的公钥对签名进行验证。RSA签名算法应用广泛，被广泛应用于电子商务、金融、网络安全等领域。  ELGAMAL签名算法： ELGAMAL签名算法是一种基于离散对数问题的数字签名算法，其基本思想是将消息和签名转化为离散对数问题，并且利用底数为素数的离散对数难题来实现签名的不可伪造性。ELGAMAL签名算法在实际应用中存在一些问题，如签名长度大、签名速度慢等。  相同点：都能用于加密，也能用于数字签名  不同点：（1）RSA的安全性依赖于大数分解，EIGamal的安全性主要依赖于p和g  （2）EIGamal体制中的随机参数的取值每次均不同，所以即使使用相同的私钥对相同的明文进行加密，每次加密后得到的签名也各不相同，而RSA体制产生密钥较麻烦，受到素数的影响，因而难以做到一次一密 | | | | | | |
| 六、教师评语  签名：  日期： | | | | | 成绩 | |

|  |
| --- |
|  |
| 椭圆曲线参数y2=x3+3x+2mod2017（即p=2017，a=3，b=2） | | |