|  |
| --- |
| **Github账号：** https://github.com/31239334/Crypto-Homework |
| **个人博客关于密码学大作业的链接：** [【现代密码学】大作业-RSA大礼包](https://blog.csdn.net/xiewenbonb/article/details/122396997)  [【现代密码学】作业一](https://blog.csdn.net/xiewenbonb/article/details/122392089) [【现代密码学】作业二](https://blog.csdn.net/xiewenbonb/article/details/122394672) |
| **题目：RSA大礼包** |
| **摘要：**  实验目的：   1. 掌握RSA加密算法中的数论知识与方法 2. 掌握几种常用的RSA攻击手段 3. 通过实践完成赛题   实验环境：  Python3.10 + Pycharm  实验题目： |
| **题目描述**  已知21组截获的加密数据，还原明文和加密传输信息，还原多者获胜。 |
| **过程**（包括背景，原理：必要的公式，图表；步骤，如有必要画出流程图，给出主要实现步骤代码）  背景：  RSA加密算法是一种非对称加密算法，是1977年由麻省理工学院的（Ron Rivest）、（Adi Shamir）和（Leonard Adleman）一起提出的。RSA就是他们三人姓氏开头字母拼在一起组成的。  实际上，1973年，在英国政府通讯总部工作的数学家克利福德·柯克斯（Clifford Cocks）在一个内部文件中提出了一个相同的算法，但他的发现被列入机密，一直到1997年才被发表。1983年麻省理工学院在美国为RSA算法申请了专利。这个专利2000年9月21日失效。由于该算法在申请专利前就已经被发表了，在世界上大多数其它地区这个专利权不被承认。RSA算法是目前使用最为广泛的非对称加密算法，这得益于它的算法原理易于理解、实现相对简单（相对于椭圆曲线公钥密码算法）。不过我们国家的非对称加密算法国家标准并不是RSA，而是基于椭圆曲线密码学的SM2。RSA算法应用非常广泛，主要可以分为如下两类：一是小数据段的加密，尤其是秘钥的传输；而是数字签名，可以说目前绝大多数的数字签名算法都是使用的RSA算法。RSA算法的出现并不是为了替代对称加密算法，通常RSA非对称加密算法要比AES、DES等对称加密算法慢的多。这是因为RSA算法的计算过程涉及到了很多大数运算，非常的耗时。RSA算法通常和AES等对称加密算法一起使用，RSA算法加密算法在这里起到了加密对称秘钥的作用，大批量数据的加密实际上使用AES等对称加密算法加密的。RSA算法的原理是由一系列的数学原理和数论知识支撑的，其中主要是模运算、费马小定理、欧拉定理和欧拉函数。RSA算法的安全性依赖于大数的相乘在计算上是简单的，但是大数的因数分解是复杂的。具体的来讲就是计算两个大质数p和q的乘积是简单的，但是想从p和q的乘积分解出p和q是非常难的。  原理：   1. 秘钥产生过程 2. 随机产生一个大素数 及模 的一个原根 3. 随机选取整数 ，计算 4. Alice的公钥为 ，私钥为 5. 加密过程（Bob将明文消息 加密成密文）    1. 随机选取一个整数 ，    2. 计算    3. 将密文 发送给Alice 6. 解密过程（Alice将密文 恢复为明文    1. 计算    2. 计算   现在我们知道了m^e % n = c是加密，c^d % n = m是解密，m就是原始数据，c是密文，公钥是n和e，私钥是n和d，所以只有n和e是公开的。加密时我们也要知道φ(n)的值，最简单的方式是用两个质数之积得到，别人想破解RSA也要知道φ(n)的值，只能对n进行因数分解，那么我们不想m被破解，n的值就要非常大，就是我们之前说的，长度一般为1024个二进制位，这样就很安全了。但是据说量子计算机(用于科研，尚未普及)可以破解，理论上量子计算机的运行速度无穷快，大家可以了解一下。  方法：   1. 费马分解法   ① 选两个保密的大素数p和q。  ② 计算n=p×q，φ(n)=(p-1)(q-1),其中φ(n)是n的欧拉函数值。  ③ 选一整数e，满足1<e<φ(n)，且gcd(φ(n),e)=1。  ④ 计算d，满足d·e≡1 mod φ(n),即d是e在模φ(n)下的乘法逆元，因e与φ(n)互素，由模运算可知，它的乘法逆元一定存在。  ⑤ 以{e,n}为公开钥,{d,n}为秘密钥。  加解密：如果明文比较长分段来处理  加密为c=m^e mod n  解密为m=c^d mod n  得到φ(n)之后的计算原理：de=1 mod φ(n)且gcd(e,φ(n))=1  即de=k\*φ(n)+1 ==> de+k\*φ(n)=1  分解思路：  直接分解：爆破（利用工具比较稳）  适用：n比较小的情况（512bit-768bit以内）  查询了解一下：http://factordb.com  利用公约数： 如果有2个n，并且2个n有相同的公约数  欧几里得算法时间复杂度为O(log n)即使是4096bit也是秒破。  def gcd(a, b):  if a < b:  a, b = b, a  while b != 0:  temp = a % b  a = b  b = temp  print a  return a  if \_\_name\_\_ == '\_\_main\_\_':  n1=input("n1:")  n2=input("n2:")  gcd(n1,n2)   1. Pollard p-1 分解法 2. 低加密指数攻击   低加密指数：当e=3时，如果明文过小，导致明文的三次方仍然小于n，那么通过直接对密文三次开方，即可得到明文。  原理：加密为c=m^e mod n  解密为m=c^d mod n  分析：如果e过小（m^e<n或者m^e<2\*n）  就可以直接对c开方。  m^e=k\*n+c  #!/usr/bin/python  # coding=utf-8  import gmpy  from Crypto.PublicKey import RSA  def calc(j):  a, b = gmpy.root(cipher + j \* N, 3)  if b > 0:  m = a  print '{:x}'.format(int(m)).decode('hex')  # pool.terminate()  # 读入公钥  with open('pubkey.pem', 'r') as f:  key = RSA.importKey(f)  N = key.n  e = key.e  # 读入密文  with open('flag.enc', 'r') as f:  cipher = f.read().encode('hex')  cipher = int(cipher, 16)  # 暴力破解  inputs = range(0, 118720000)  result = []  map(calc, inputs)  print len(result)   1. 公共模数攻击   说明：如果在RSA的使用中使用了相同的模n对相同的明文m进行了加密，那么就可以在不分解n的情况下还原出明文m的值。(e\_1和e\_2互素)  c\_1=m^(e\_1)mod n  c\_2=m^(e\_2)mod n  需要：密文c\_1,c\_2 并且n相同.  结果：  c\_1^{s\_1}\*c\_2^{s\_2}=m mod n  s\_1\*e\_1+s\_2\*e\_2=1  证明：gcd (e\_1,e\_2)=1  即存在s\_1 ，s\_2使得：  s\_1\*e\_1+s\_2\*e\_2=1  又因为： c\_1= m^{e\_1}mod n  c\_2= m^{e\_2} mod n  通过代入化简可以得出：  c\_1^{s\_1}\*c\_2^{s\_2}= m mod n  代码：  #coding=utf-8  import gmpy2  from libnum import n2s,s2n  n=  e1=17  e2=65537  fo1=open("flag.enc1",'rb')  fo2=open("flag.enc2","rb")  datafo1=fo1.read()  message1=s2n(datafo1)  fo1.close()  datafo2=fo2.read()  message2=s2n(datafo2)  gcd,s,t=gmpy2.gcdext(e1,e2)  print message1  print gcd,s,t  plain=gmpy2.powmod(message1 ,s,n)\*gmpy2.powmod(message2,t,n)%n  print plain  print n2s(plain)   1. 因数碰撞法   检测截获的加密数据中是否存在模数N 间有相同的质因素，可快速分解出这些N 的另一个质因数，适用于当加密方选取的模数有一个相同的质因子。  原理和算法流程：  1）对截获的21 个模数N 两两求最大公因数，排除模数相同的情况，设检测的模数分别为  N1 , N2  2）当gcd(N1 , N2)> 1 时，利用得到的最大公因数分别求得另外一个质因数，再进行明文的破密.   1. 低加密指数攻击   检测截获的加密数据中是否存在加密指数较小，且加密明文相同的密文帧  适用对象/情况：当n 个加密方选取的加密指数e 较小，且用以加密相同的明文，依照题意，满足1024\*n> =512\*e  原理和算法流程：在截获的所有加密信息帧中，设有n 个选用了相同的低加密指数e,设存在：  1 ）对k 个组合尝试用中国剩余定理求解出x= memod Nk  2）开根号, 求解出m  3）若以上步骤顺利进行(即求解出的x 能开e 次根), 且解出的明文符合题目所给格式要求且具有合理含义，则破解成功，反之，说明，所选取的破解组合没有选用相同的明文，无法使用低加密指数破解。   1. 猜测明文攻击   根据已有明文信息，猜测明文，还原  通过上面几种方式获得的一些明文信息，通过合适的顺序调整后，在浏览器中搜索，基本可得相符的语句： My secret is a famous saying of Albert Einstein. That is “Logic will get you from A to B. Imagination will take you everywhere.”  当然，这种解密方式不适用于所有情况。 |
| **总结（完成心得与其它，主要自己碰到的问题和解决问题的方法）**  对RSA的原理和攻击方式有了进一步的认识，了解到了一些RSA的攻击方式，对RSA加解密及其相关数学原理和实践能力进一步加深。 |
| **代码：**  1、 Chinese\_remainder\_theorem.py  import gmpy2  from functools import reduce  import itertools  import os  def jugde\_prime(num\_list): # 判断是否两两互素  bnum\_pairs=list(itertools.combinations(num\_list.values(),2) ) # 两两组合,取字典中键值  for bnum1,bnum2 in bnum\_pairs:  if gmpy2.gcd(bnum1,bnum2)!=1: # 求最大公因数，是1则互素  return False  return True  def num\_list\_inversion(num\_list):  sol\_list=[]  # 求所有模数的累积  M=reduce(lambda x,y:x\*y,num\_list.values())  for a in num\_list:  Mj=M//num\_list[a]  iverm=gmpy2.invert(Mj,num\_list[a]) # iver=Mi^(-1) (modmi)  xj=gmpy2.t\_mod(Mj\*iverm\*a,M) # xj=Mj\*Mj^(-1)\*aj (mod M)  sol\_list.append(xj)  return sol\_list,M  def Chinese\_Remainder\_Theory(num\_list): # num\_list:a1,a2,a3,m1,m2,m3  if not jugde\_prime(num\_list):  print("m1,m2,m3 不满足两两互素，不能利用中国剩余定理！")  return None  sol\_list,M=num\_list\_inversion(num\_list)  x=reduce(lambda x,y:gmpy2.t\_mod(x+y,M),sol\_list) # x=(1-k)+xj(mod M)  #print("用中国剩余定理求得解为：\nx=%d(mod%d)"%(x,M))  '''  for a in num\_list:  index=list(num\_list.keys()).index(a)+1  print('a%d='%index,a)  print('x(mod m%d)='%index,divmod(x,num\_list[a])[1])  return x,M  '''  return x,M  def test(dir):  '''  system\_of\_equations(input): a1:m1,a2:m2,a3:m3  '''  for test\_f in sorted(os.listdir(dir),key=lambda x:int(x[:-4])):  print('\n\ntesting %s...'%test\_f)  num\_list=open(dir+test\_f,'r').read().strip().split('\n')  if len(num\_list)!=6:  print('Invaid input!')  continue  num\_list=[gmpy2.mpz(x) for x in num\_list] # 转化为大整数形式，有利于提升后面计算速度  system\_of\_equations=dict(zip(num\_list[:3],num\_list[3:]))  Chinese\_Remainder\_Theory(system\_of\_equations)  if \_\_name\_\_=='\_\_main\_\_':  #dir=r'/Users/apple/Downloads/20个数据/1'  #test(dir)  dir=r'/Users/apple/Downloads/1.txt'  num\_list=open(dir,'r').read().strip().split('\n')  num\_list=[gmpy2.mpz(x) for x in num\_list] # 转化为大整数形式，有利于提升后面计算速度  system\_of\_equations=dict(zip(num\_list[:3],num\_list[3:]))  print(system\_of\_equations)  Chinese\_Remainder\_Theory(system\_of\_equations)  2、 Algorithm\_Base.py（ Pollard p- 1 分解法）  # coding=gbk  import gmpy2  def pollard\_p\_q(N):  B=gmpy2.mpz(gmpy2.factorial(2\*\*14))  a=0;b=0;det=0  a\_=gmpy2.iroot(N,2)[0]+1  while(det<B):  a=pow((a\_+det),2)-N  if gmpy2.isqrt(a):  b=gmpy2.isqrt(a)  break  det+=1  p=a\_+det+b  return p  def ppl(N):  ''' Pollard p-1分解法 , 适用于p-1或q-1能够被小素数整除的情况 '''  B=gmpy2.mpz(gmpy2.exp2(20))  a=2  for i in range(2,B+1):  a=gmpy2.powmod(a,i,N)  d=gmpy2.gcd(a-1,N)  if d>=2 and d<=(N-1):  q=N//d  N=q\*d  return d  3、 RSA\_Challenge.py  import gmpy2  import os  import binascii  import itertools  from Algorithm\_Base import pollard\_p\_q,ppl  import math  from Chinese\_remainder\_theorem import Chinese\_Remainder\_Theory  import time  def load\_file(dir): # 根据信息格式加载文件  f=open(dir,'r+')  data=f.read().strip()  #print('len of the data:',len(data))  N=gmpy2.mpz(int('0x'+data[:256],16))  e=gmpy2.mpz(int('0x'+data[256:256\*2],16))  c=gmpy2.mpz(int('0x'+data[256\*2:],16))  return N,e,c  def RSA\_Decryption(c,e,p,q,N):  d=gmpy2.invert(e,gmpy2.mpz((p-1)\*(q-1)))  m=gmpy2.powmod(c,d,N)  # print('hex m:',str(hex(m))[2:])  plaintext=binascii.unhexlify((str(hex(m))[2:]).encode())  return plaintext  def Fermat\_Decomposition(N): # 费马分解法-用来攻击参数选择不当的RSA密文  """ 当N=p\*q选择不当，即p,q过于接近时,可以较快的暴力破解出p,a的值,从而破密 """  p=pollard\_p\_q(N)  if gmpy2.is\_prime(p):  print('p:',p)  q=N//p  return p,q  else:  return None  def Load\_Frames(dir):  Frame\_Info={}  for test\_f in sorted(os.listdir(dir),key=lambda x:int(x[5:])):  #print('\n\nLoading information about %s...'%test\_f)  N,e,c=load\_file(dir+test\_f)  Frame\_Info.update({int(test\_f[5:]):{'N':N,'c':c,'e':e}})  #print('N:',N)  #print('c:',c)  #print('e:',e)  return Frame\_Info  def Test\_Fermat\_Decomposition(Frame\_Info):  for frame\_num in Frame\_Info:  #print('\n\ntesting %d...'%frame\_num)  cipher\_key=Fermat\_Decomposition(Frame\_Info[frame\_num]['N'])  if cipher\_key==None:  continue  p,q=cipher\_key  m=RSA\_Decryption(Frame\_Info[frame\_num]['c'],Frame\_Info[frame\_num]['e'],p,q,Frame\_Info[frame\_num]['N'])  print('Crack Successfully!\nFrame %d:\n'%frame\_num,m)  def Test\_Low\_Encryption\_Index\_Attack(e,frame\_index\_list,Frame\_Info):  n=math.ceil(512\*e/1024) # 最小可以用低指数攻击的n  for k in range(n,e+1):  plaintext=[]  combs=itertools.combinations(frame\_index\_list,k)  for com in combs:  c=[Frame\_Info[fn]['c'] for fn in com]  N=[Frame\_Info[fn]['N'] for fn in com]  c\_N=dict(zip(c,N))  m,M=Chinese\_Remainder\_Theory(c\_N)  if gmpy2.iroot(m,k)[1]:  plaintext.append(gmpy2.iroot(m,k)[0])  if len(set(plaintext))==1:  # print(set(plaintext))  plaintext=binascii.unhexlify(str(hex(plaintext[0]))[2:])  for fn in frame\_index\_list:  print('Frame %d '%fn,end='')  print(':\n',plaintext[-8:])  return plaintext  for fn in frame\_index\_list:  print('Frame %d '%fn,end='')  print('\nDecrypt Failed!')  def egcd(a,b): # 扩展欧几里得算法  if a==0:  return (b,0,1)  else:  g,y,x=egcd(b%a,a)  return (g,x-(b//a)\*y,y)  def Test\_Common\_Modulus\_Index\_Attack(Frame\_Info): # 公共模攻击  Ns=[Frame\_Info[fn]['N'] for fn in Frame\_Info]  frame\_index\_list=[]  for N in set(Ns):  if Ns.count(N)>=2:  frame\_index\_list.append([index for index in Frame\_Info if Frame\_Info[index]['N']==N])  print('frame\_index\_list:',frame\_index\_list)  for common\_N\_frames in frame\_index\_list:  for cNf in itertools.combinations(common\_N\_frames,2):  e1 = Frame\_Info[cNf[0]]['e']  e2 = Frame\_Info[cNf[1]]['e']  c1 = Frame\_Info[cNf[0]]['c']  c2 = Frame\_Info[cNf[1]]['c']  N = Frame\_Info[cNf[0]]['N']  \_,s,t=egcd(e1,e2)  if s<0:  s=-s  c1=gmpy2.invert(c1,N)  elif t<0:  t=-t  c2=gmpy2.invert(c2,N)  m=pow(c1,s,N)\*pow(c2,t,N)%N  plaintext=binascii.unhexlify(str(hex(m))[2:])  print('Frame %d ; Frame %d'%(cNf[0],cNf[1]))  print('plaintext:',plaintext[-8:])  def Factor\_Collision\_Attack(Frame\_Info):  N=[Frame\_Info[fn]['N'] for fn in Frame\_Info]  for N\_com in itertools.combinations(N,2):  if N\_com[0]==N\_com[1]: # 在公共模数攻击里面已完成，而且相等的模数也不能用因子碰撞  continue  if gmpy2.gcd(N\_com[0],N\_com[1])!=1:  p=gmpy2.gcd(N\_com[0],N\_com[1])  q1=N\_com[0]//p  q2=N\_com[1]//p  index1=N.index(N\_com[0])  index2=N.index(N\_com[1])  c1=Frame\_Info[index1]['c']  c2=Frame\_Info[index2]['c']  e1=Frame\_Info[index1]['e']  e2=Frame\_Info[index2]['e']  plaintext1=RSA\_Decryption(c1,e1,p,q1,N\_com[0])  plaintext2=RSA\_Decryption(c2,e2,p,q2,N\_com[1])  print('Frame %d:\n'%index1,plaintext1[-8:])  print('Frame %d:\n'%index2,plaintext2[-8:])  def Test\_Pollard\_p\_1(Frame\_Info):  for fn in Frame\_Info:  #print('Testing Frame %d...'%fn)  N=Frame\_Info[fn]['N']  time0=time.time()  p=ppl(N)  if p==1:  #print('p=%d ==> The attack cannot be applied to the Frame.'%p)  continue  q=Frame\_Info[fn]['N']//p  c=Frame\_Info[fn]['c']  e=Frame\_Info[fn]['e']  plaintext=RSA\_Decryption(c,e,p,q,N)  print('Frame %d:\n'%fn,plaintext[-8:])  if \_\_name\_\_=='\_\_main\_\_':  # dir\_sample=r'/Users/apple/Downloads/密码挑战赛赛题三/附件3-1（加密案例）/'  dir\_test=r'/Users/apple/Desktop/课程/对称密码/实验报告/18069100135-陈波妃-对称密码分析第四次实验作业/Crypto Challenge Three/附件3-2（发布截获数据）/'  Frame\_Info=Load\_Frames(dir\_test)  #print('Frame Information:\n',Frame\_Info)  # 费马因数分解法(测试p,q选取过于接近的不安全情况)  print("Fermat\_Decomposition...")  Test\_Fermat\_Decomposition(Frame\_Info)  print("============================================================================================")  '''  Crack Successfully!  Frame 10:  b'\x98vT2\x10\xab\xcd\xef\x00\x00\x00\x08\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00  \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00  \x00\x00\x00\x00\x00\x00will get'  去填充:will get  '''  #e=[(fn,Frame\_Info[fn]['e']) for fn in Frame\_Info]  #print('es:',e)    # 低加密指数攻击  print("Low\_Encryption\_Index\_Attack...")  Test\_Low\_Encryption\_Index\_Attack(3,[7,11,15],Frame\_Info)  print("============================================================================================")  '''  Frame 7 Frame 11 Frame 15  Decrypt Failed!  '''  # Frame 3,8,12,16,20  print("Low\_Encryption\_Index\_Attack...")  Test\_Low\_Encryption\_Index\_Attack(5,[3,8,12,16,20],Frame\_Info)  print("============================================================================================")  '''  Frame 3 Frame 8 Frame 12 Frame 16 Frame 20 :  b't is a f'  '''  # 公共模数攻击  print("Common\_Modulus\_Index\_Attack...")  Test\_Common\_Modulus\_Index\_Attack(Frame\_Info)  print("============================================================================================")  '''  frame\_index\_list: [[0, 4]]  Frame 0 ; Frame 4  plaintext: b'My secre'  '''    # 因数碰撞攻击  print("Factor\_Collision\_Attack...")  Factor\_Collision\_Attack(Frame\_Info)  print("============================================================================================")  '''  Frame 1:  b'. Imagin'  Frame 18:  b'm A to B'  '''    # Pollard p-1分解法  print("Pollard\_p\_1 Decomposition...")  Test\_Pollard\_p\_1(Frame\_Info)  print("============================================================================================")  '''  Frame 2:  b' That is'  Frame 6:  b' "Logic '  Frame 19:  b'instein.'  ''' |
| **参考文献**  <https://www.52pojie.cn/thread-653446-1-1.html>  [RSA原理\_baidu\_31405631的博客-CSDN博客\_rsa的工作原理](https://blog.csdn.net/baidu_31405631/article/details/107243315?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522164173664416780269840595%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=164173664416780269840595&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_click~default-1-107243315.pc_search_insert_ulrmf&utm_term=RSA%E5%8E%9F%E7%90%86&spm=1018.2226.3001.4187)  [RSA攻击方法整理\_九层台-CSDN博客\_rsa攻击](https://blog.csdn.net/qq_38204481/article/details/83189041?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522164173728416780271529728%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=164173728416780271529728&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-83189041.pc_search_insert_ulrmf&utm_term=RSA%E6%94%BB%E5%87%BB&spm=1018.2226.3001.4187)  <https://blog.csdn.net/weixin_46395886/article/details/114434642> |

**简要描述要求，目的等**

关于密码学编程大作业的说明：

1. 密码学编程大作业三次，前期课堂两次大多数同学已经在雨课堂（0930和1024）提交如果没有提交请这次补上。
2. 要求提交的大作业以给出成绩，我们只交一次纸质版，内容为最后一次大作业RSA大礼包。请建立自己的代码托管账号，Github或Coding或其他托管平台均可，建立合理的文件目录托管代码，请清晰命名，给出必要注释；
3. 请建立自己的技术博客或者其它记录载体，简单记录三次实验内容，所遇到的问题以及心得。
4. 电子版大作业提交至 63307507@qq.com，邮件命名为“姓名\_学号\_密码学大作业”，提交pdf版附件命名“姓名\_学号\_密码学大作业”。 提交时间1月9日（本周末）晚23:00前。