|  |
| --- |
| **Github账号：**acetyl-lwx |
| **个人博客关于密码学实验的链接：**https://www.cnblogs.com/Yanami-8/articles/18592679 |
| **实验题目：**现代密码学实验四 |
| **实验摘要：**  本次实验涉及RSA算法，并且使用了几种针对RSA加密的常见攻击方法。 |
| **题目描述**  2016 年全国高校密码数学挑战赛赛题三。  有人制作了一个RSA加解密软件。已知该软件发送某个明文的所有参数和加密过程的全部数据（加密案例文件详见附件3-1）。Alice使用该软件发送了一个通关密语，且所有加密数据已经被截获，请问能否仅从加密数据恢复该通关密语及RSA体制参数？如能请给出原文和参数，如不能请给出已恢复部分并说明剩余部分不能恢复的理由？ |
| **过程**  思路  本问题是一个传统的RSA密码体制破译问题。题目中已知明文的加密规则，并截获了所有的加密指数e、模数N和密文c，要求据此求出解密指数d、素数p、q并最终恢复全部的明文m。  本题中，模数N的长度为1024bit，就目前的计算机水平来说，1024bit长度的模数是基本安全的。直接对1024bit的模数进行因数分解在操作可行性上存在很大困难，然而对于某些存在不安全的RSA加解密体制参数，我们可以用特定的攻击方法，针对其中隐藏的安全缺陷进行破译。  根据题目中加密帧数据的格式分离出每个片段的模数N、加密指数e和密文c。其中，明文分成8个字符后进行加密，每一分片较短，符合猜测明文攻击的条件；有三组加密指数e同为3，有五组加密指数e同为5，存在明文片段相同的可能性，可以尝试公共模数攻击法对其进行破解；素数p、q中至少有一个是由随机数发生器生成的，若能找到随机数生成的规律，就可以破解全部加解密参数。  具体解题方法  1、费马分解法  当p与q十分靠近时,费马分解法可以有效地分解N,这里不介绍具体算法。根据该原理，对所有21个分片进行攻击后，Frame10在较短时间内被成功破解。  完整代码如下：  import gmpy2  def factor(n):      a=gmpy2.iroot(n,2)[0]      while True:          a+=1          b2=a\*a-n          if gmpy2.is\_square(b2):              b2=gmpy2.mpz(b2)          b,xflag=gmpy2.iroot(b2,2)          assert xflag          return (a-b,a+b)    def hex\_to\_char(hex\_str):      return bytes.fromhex(hex\_str).decode('utf-8')  with open("Frame10") as file:      message = file.read()      n=int(message[:256],16)      e=int(message[256:512],16)      c=int(message[512:],16)  file.close()  p,q=factor(n)  print("p=",hex(p))  print("q=",hex(q))  fai=(p-1)\*(q-1)  d=gmpy2.invert(e,fai)  print("d=",hex(d))  m=gmpy2.powmod(c,d,n)  print("m=",hex(m))  plaintext=hex(m)[-16:]  print(hex\_to\_char(plaintext))  2、Pollard p-1分解法  RSA加密算法中，素数p和q应该满足：和至少有一个素因子大于，否则，要分解的整数N有一个素因数p，且(p-1)有小的素因数时就可以利用Pollard p-1分解法有效分解模数N，从而得到p。  对所有分片进行攻击后，成功破解了Frame2、Frame6和Frame19。  完整代码：  import gmpy2  def pollard(N):      a = 2      B = 2      while True:          a = gmpy2.powmod(a, B, N)          res = gmpy2.gcd(a-1, N)          if res != 1 and res != N:              q = N // res              return res, q          B += 1  def hex\_to\_char(hex\_str):      return bytes.fromhex(hex\_str).decode('utf-8')  with open("Frame19") as file:      message = file.read()      n=int(message[:256],16)      e=int(message[256:512],16)      c=int(message[512:],16)  file.close()  p,q=pollard(n)  print("p=",hex(p))  print("q=",hex(q))  fai=(p-1)\*(q-1)  d=gmpy2.invert(e,fai)  print("d=",hex(d))  m=gmpy2.powmod(c,d,n)  print("m=",hex(m))  plaintext=hex(m)[-16:]  print(hex\_to\_char(plaintext))  3、低加密指数攻击  在设计算法时为了加速计算，常常使用较小的加密指数e，但是这种做法存在很大的安全隐患。如果相同的消息使用同样的加密指数加密后发送给不同的接收者，则该明文消息可以被非常有效的恢复。该攻击方法基于中国剩余定理。  低加密指数攻击法适用于加密密钥相同且模数互素的数，通过对数据的观察，我们发现Frame7、Frame11、Frame15 和Frame3、Frame8、Frame12、Frame16、Frame20这两组数分别符合低加密指数攻击的条件。  对Frame7、Frame11和Frame15进行攻击，无论是假设三帧有相同明文还是其中两帧有相同明文，均无法得到有意义的明文，攻击失败。  对Frame3、Frame8、Frame12、Frame16、Frame20进行攻击，假设五帧都有相同明文，利用中国剩余定理求解，能够得到有意义的明文。  完整代码如下：  import gmpy2  #求最大公约数  def gcd(a, b):      if b == 0:          return a      else:          return gcd(b, a % b)  #求Mi  def Get\_Mi(m\_list, m):      M\_list = []      for mi in m\_list:          M\_list.append(m // mi)      return M\_list  #求Mi的逆元  def Get\_resMi(M\_list, m\_list):      resM\_list = []      for i in range(len(M\_list)):          resM\_list.append((Get\_ni(M\_list[i], m\_list[i])[0]+m\_list[i])%m\_list[i])      return resM\_list  #拓展欧几里得算法  def Get\_ni(a, b):      if b == 0:          x = 1          y = 0          q = a          return x, y, q      ret = Get\_ni(b, a % b)      x = ret[0]      y = ret[1]      q = ret[2]      temp = x      x = y      y = temp - a // b \* y      return x, y, q  #求解结果  def result(a\_list, m\_list):      for i in range(len(m\_list)):          for j in range(i + 1, len(m\_list)):              if 1 != gcd(m\_list[i], m\_list[j]):                  print("不能直接使用中国剩余定理")                  return      m = 1      for mi in m\_list:          m \*= mi      Mi\_list = Get\_Mi(m\_list, m)      Mi\_inverse = Get\_resMi(Mi\_list, m\_list)      x = 0      xi\_list = []      for i in range(len(a\_list)):          xi = a\_list[i] \* Mi\_list[i] \* Mi\_inverse[i]          xi\_list.append(xi)          x += xi          x %= m      a,b=gmpy2.iroot(x,5)      if b==1:          x=a      return x  def hex\_to\_char(hex\_str):      return bytes.fromhex(hex\_str).decode('utf-8')  #读取数据  n=0  a\_list=[]  m\_list=[]  with open("frame3") as file:      message = file.read()      n=int(message[:256],16)      e=int(message[256:512],16)      c=int(message[512:],16)      a\_list.append(c)      m\_list.append(n)  file.close()  with open("frame8") as file:      message = file.read()      n=int(message[:256],16)      e=int(message[256:512],16)      c=int(message[512:],16)      a\_list.append(c)      m\_list.append(n)  file.close()  with open("frame12") as file:      message = file.read()      n=int(message[:256],16)      e=int(message[256:512],16)      c=int(message[512:],16)      a\_list.append(c)      m\_list.append(n)  file.close()  with open("frame16") as file:      message = file.read()      n=int(message[:256],16)      e=int(message[256:512],16)      c=int(message[512:],16)      a\_list.append(c)      m\_list.append(n)  file.close()  with open("frame20") as file:      message = file.read()      n=int(message[:256],16)      e=int(message[256:512],16)      c=int(message[512:],16)      a\_list.append(c)      m\_list.append(n)  file.close()  plaintext=hex(result(a\_list, m\_list))  print(hex\_to\_char(plaintext[-16:]))  4、公共模数攻击  RSA加密中的四个参数{d,p,q,ϕ(n)}是同等重要的，任何一个参数泄露都会给其余三个的安全性带来威胁。但是，如果在RSA加密中使用相同的模数，那么有可能在不知道四个参数知识的情况下，攻击得到RSA加解密体制参数。  通过观察数据，我们了解到：Frame0与Frame4有相同的模数。猜测这两组数据是由同一明文信息加密得到的，则可以尝试公共模数攻击法。经实验，攻击成功。  完整代码如下：  #扩展欧几里得算法  def ext\_gcd(a, b):      if b == 0:          return 1, 0, a      else:          x, y, gcd = ext\_gcd(b, a % b)          x, y = y, (x - (a // b) \* y)          return x, y, gcd  def hex\_to\_char(hex\_str):      return bytes.fromhex(hex\_str).decode('utf-8')  with open("Frame0") as file:      message=file.read()      n\_1=int(message[:256],16)      e\_1=int(message[256:512],16)      c\_1=int(message[512:],16)  file.close()  with open("Frame4") as file:      message=file.read()      n\_2=int(message[:256],16)      e\_2=int(message[256:512],16)      c\_2=int(message[512:],16)  file.close()  x,y,g=ext\_gcd(e\_1,e\_2)  plaintext=pow(c\_1,x,n\_1)\*pow(c\_2,y,n\_2)  plaintext=plaintext%n\_1  print(hex(plaintext))  print(hex\_to\_char(hex(plaintext)[-16:]))  5、模不互素  由于模数的因子p都是由同一随机数发生器产生的，如果随机数发生器由于在启动时熵值不高，就有可能发生两个模数有共同的因子p，这样只需要用欧几里得算法求两两模数的最大公因数就能够分解模数。经过枚举，发现Frame1和Frame18的模数有公共因子，因而可以据此对这两帧的模数进行分解，最后破解明文。  完整代码如下：  import gmpy2  def gcd(a, b):      if b == 0:          return a      else:          return gcd(b, a % b)  def hex\_to\_char(hex\_str):      return bytes.fromhex(hex\_str).decode('utf-8')  with open("Frame1") as file:      message=file.read()      n\_1=int(message[:256],16)      e\_1=int(message[256:512],16)      c\_1=int(message[512:],16)  file.close()  with open("Frame18") as file:      message=file.read()      n\_2=int(message[:256],16)      e\_2=int(message[256:512],16)      c\_2=int(message[512:],16)  file.close()  p=gcd(n\_1,n\_2)  q\_1=n\_1//p  phi\_1=(p-1)\*(q\_1-1)  d\_1=gmpy2.invert(e\_1,phi\_1)  c\_1=pow(c\_1,d\_1,n\_1)  ciphertext\_1=hex(c\_1)  print(ciphertext\_1)  print(hex\_to\_char(ciphertext\_1[-16:]))  q\_2=n\_2//p  phi\_2=(p-1)\*(q\_2-1)  d\_2=gmpy2.invert(e\_2,phi\_2)  c\_2=pow(c\_2,d\_2,n\_2)  ciphertext\_2=hex(c\_2)  print(ciphertext\_2)  print(hex\_to\_char(ciphertext\_2[-16:]))  6、猜测明文  通过已知片段猜测明文，然后枚举明文验证。  最终，全部明文信息破译成功，将所有分片的明文信息按照通信序号连接起来得到通关密语：My secret is a famous saying of Albert Einstein. That is "Logic will get you from A to B. Imagination will take you everywhere." |
| **总结**  本次实验涉及了对RSA加密算法的多种攻击，告诉我们在设计RSA算法时需要避免有风险的设计缺陷。 |
| **参考文献**  <https://www.freebuf.com/articles/database/290623.html> |