|  |
| --- |
| **Github账号：observer-297** |
| **个人博客关于密码学实验的链接：**[**observer-297/cryptography**](https://github.com/observer-297/cryptography) |
| **实验题目：RSA大礼包** |
| **实验摘要：**  **RSA算法是目前应用最广泛的公钥密码体制之一，其设计基于大整数分解困难性，因而以高安全性著称。本文对21组加密数据进行了分类与分析，并通过五种方法成功破解了大部分密文。对于剩余未破解部分，通过英语语义分析进行推测和验证，最终还原了全部明文信息。本文所采用的破解方法大多基于RSA密文破解的常规思路，将在正文中逐一展开分析。** |
| **题目描述**  **题目来源于2016年全国高校密码数学挑战赛的赛题三——“RSA加密体制破译”。题目描述了一种RSA加密软件（由于设计中存在某些不规范之处，导致其存在被攻击的可能性）。已知该软件在加密某个明文时，所有参数和加密过程的中间数据均被截获（包括已知明密文攻击的条件以及加密过程中的中间结果）。Alice 使用该软件发送了一段通关密语，而相关的加密数据已被完全获取。挑战者需要运用RSA的各种攻击手段，仅通过这些加密数据恢复出通关密语以及RSA体制的相关参数。**  **RSA算法仅需选择5个核心参数（N, p, q, e, d），实现简单且安全性较高。理论上，若N为1024位，其加密强度足够安全，但如果参数选择不当，可能会引发安全漏洞。题目提供了该加密软件在21个帧（Frame）中的加密数据，要求挑战者通过分析这些数据，成功还原21个帧的明文信息，并最终恢复通关密语。** |
| **过程**  **过程中我们使用公共模数攻击，因数碰撞，低加密指数攻击(小e)，以及费马分解和p-1分解这五种攻击方法**  **我们根据数据帧中给出的信息来选择使用什么攻击方法**  **帧数据的数据格式如下，其中数据都是 16 进制表示，结构如下 1024bit模数N | 1024bit加密指数e | 1024bit密文m e mod N。**  **首先提取出数据帧的有效信息，根据这些信息进行分析选取攻击方法。**  # 模数集合  n = []  # 公钥指数集合  e = []  # 密文集合  c = []  # 明文集合  m = {}  # 已解密的密文集合  solved = []  # 文件名集合  filenames = [f'RSA\\data\\Frame{i}' for i in range(21)]  # 读取每个文件，解析 n、e 和 c 的值  for i in range(21):      with open(filenames[i], 'r') as f:          data = f.read()          # 将字符串转为十六进制，再转换为整数          n.append(int(data[:256], 16))       # 模数 n          e.append(int(data[256:512], 16))   # 公钥指数 e          c.append(int(data[512:], 16))      # 密文 c  # 输出 e 的值  for i in range(21):      print(f'e[{i}] = {e[i]}')  # 输出 n 的值  for i in range(21):      print(f'n[{i}] = {n[i]}')  # 输出 c 的值  for i in range(21):      print(f'c[{i}] = {c[i]}')  **这样我们就提取出了所有模数N、加密指数e及密文c。**  **1．公共模数攻击**    **Frame0与4的N和加密片段均相同，可以共模攻击**  import gmpy2  import binascii  # 扩展欧几里得算法，用于求解 ax + by = gcd(a, b)  def exgcd(a, b):      if b == 0:          return 1, 0, a      else:          x, y, r = exgcd(b, a % b)          x, y = y, x - (a // b) \* y          return x, y, r  # 公共模数攻击  def same\_mod\_attack(n, e1, e2, c1, c2):      # 使用扩展欧几里得算法求解 e1 和 e2 的线性组合系数      x, y, r = exgcd(e1, e2)      # 如果 x 或 y 为负数，则需要求模逆元      if x < 0:          x = -x          c1 = gmpy2.invert(c1, n)      if y < 0:          y = -y          c2 = gmpy2.invert(c2, n)      # 使用公式 m = (c1^x \* c2^y) % n 计算明文      m = pow(c1, x, n) \* pow(c2, y, n) % n      # 将明文转换为十六进制字符串      m = hex(m)[2:]  # 去掉 "0x"      # 将十六进制字符串转换为明文字符串      m = binascii.unhexlify(m)[-8:]  # 只取最后 8 个字节（明文部分）      return m  if \_\_name\_\_ == '\_\_main\_\_':      # 公共模数攻击      for i in range(21):          for j in range(i + 1, 21):              if n[i] == n[j]:  # 如果两个帧的模数相同                  m2 = same\_mod\_attack(n[i], e[i], e[j], c[i], c[j])  # 执行公共模数攻击                  sloved[f'Frame{i}'] = m2  # 存储解密结果                  sloved[f'Frame{j}'] = m2  # 两个帧的明文相同      # 输出已解密的结果      print(sloved)  **2.模数公因数攻击**  **攻击原理**  **当存在两个公钥的 N 不互素时，我们显然可以直接对这两个数求最大公因数，然后直接获得 p，q，进而获得相应的私钥**  **Frame1、Frame18采用该种攻击方法**  **直接计算两个模数的gcd即可成功分解。**  **3．低指数广播攻击**    **Frame3、Frame8、Frame12、Frame16、Frame20采用该种攻击方法，e为3或者5，不断累加n直至开出整数即可。**  #中国剩余定理  def chinese\_remainder\_theorem(backup):          #计算N的乘积          N=1          for a,n in backup:                  N\*=n          #计算Ni          Ni=[]          for a,n in backup:                  Ni.append(N//n)          #计算Ni的模逆元          Ni\_inverse=[]          for i in range(0,len(Ni)):                  Ni\_inverse.append(gmpy2.invert(Ni[i],backup[i][1]))          #计算x          x=0          for i in range(0,len(Ni)):                  x+=backup[i][0]\*Ni[i]\*Ni\_inverse[i]          x=x%N          return x,N  #低指数3  def low\_exponent\_attack3():          frame\_range=[7,11,15]          backup=[]          for i in frame\_range:                  backup.append([c[i],n[i]])          x,N=chinese\_remainder\_theorem(backup)          #开三次方根          m=gmpy2.iroot(x,3)[0]          m=hex(m)[2:]#去掉0x          m=binascii.unhexlify(m)[-8:]#hex->str          sloved['Frame7']=m          sloved['Frame11']=m          sloved['Frame15']=m  #低指数5  def low\_exponent\_attack5():          frame\_range=[3,8,12,16,20]          backup=[]          for i in frame\_range:                  backup.append([c[i],n[i]])          x,N=chinese\_remainder\_theorem(backup)          #开五次方根          m=gmpy2.iroot(x,5)[0]          m=hex(m)[2:]#去掉0x          m=binascii.unhexlify(m)[-8:]#hex->str          sloved['Frame3']=m          sloved['Frame8']=m          sloved['Frame12']=m          sloved['Frame16']=m          sloved['Frame20']=m  **4.费马分解法**    **当p和q非常接近时，在根号N的附近进行枚举搜索即可成功分解N。**  **5.** **Pollard p-1分解法**    import gmpy2  import binascii  # Pollard's p-1 算法，用于分解整数 n  def pollard\_p\_1(n):      b = 2\*\*20  # 设置上界      a = 2      # 初始值      for i in range(2, b + 1):          # 计算 a^i % n          a = gmpy2.powmod(a, i, n)          # 计算 gcd(a - 1, n)          d = gmpy2.gcd(a - 1, n)          # 如果找到非平凡因子，则返回          if d != 1 and d != n:              return d      return None  # 未找到因子时返回 None  # 使用 Pollard's p-1 算法处理特定帧数据  def pollard\_data(n):      frame\_range = [2, 6, 19]  # 指定需要处理的帧索引      for i in frame\_range:          temp\_n = n[i]          temp\_c = c[i]          temp\_e = e[i]            # 使用 Pollard's p-1 算法分解模数 n          p = pollard\_p\_1(temp\_n)          if p is None:              print(f"Frame {i}: Pollard's p-1 failed to factorize n.")              continue          q = temp\_n // p  # 计算另一个因子 q          phi = (p - 1) \* (q - 1)  # 计算欧拉函数 φ(n)            # 计算私钥 d          d = gmpy2.invert(temp\_e, phi)            # 解密密文          m = pow(temp\_c, d, temp\_n)            # 将解密后的明文转换为字符串          m = hex(m)[2:]  # 去掉 "0x"          m = binascii.unhexlify(m)[-8:]  # 转换为明文字符串          sloved[f'Frame{i}'] = m  # 存储解密结果  if \_\_name\_\_ == '\_\_main\_\_':      pollard\_data(n)      print(sloved)  **Frame10可以采用该种攻击方法**  **最终我们可以得到如下数据**    **我们可以得到** **My secret is a fXXXinstein. That is "Logic will getXXXm A to B. ImaginXXX**  **可以直接在搜索引擎上进行搜索即可成功的得到**    **我们便可以得到全部的明文信息。** |
| **总结**  **本题又叫做RSA大礼包，因为其涉及到了多种RSA加密算法的相关攻击又不涉及到过度复杂的数学推导，适合初学者尝试这些攻击方法，并注意到RSA算法需要避免的一些参数问题。** |
| **参考文献（包括参考的书籍，论文，URL等，很重要）**  **[ctf wiki]**[**公钥指数相关攻击 - CTF Wiki**](https://ctf-wiki.org/crypto/asymmetric/rsa/rsa_e_attack/)  **[Twenty Years of Attacks on the RSA Cryptosystem] https://www.ams.org/notices/199902/boneh.pdf** |