|  |
| --- |
| **Github账号：** shal10w |
| **个人博客关于密码学大作业的链接：** |
| **题目：RSA大礼包** |
| **摘要：**  在密码学中，rsa是目前应用最广的一种公钥密码体制，但是不规范的使用rsa会导致很多的安全问题，本次实验就是通过分析一些不规范使用的rsa，编写攻击程序进行破解，从而锻炼我们分析算法能力于程序编写能力，并加深对rsa的理解。 |
| **题目描述**  题目给出了20个rsa加密后的密文帧于对应的公钥。需要我们通过分析公钥的缺陷来恢复明文。 |
| **过程** 首先我们用现有的分解大整数的方法对所有的模数进行测试  费马分解法  费马分解法用于p与q相近的n  注意到  因此(p+q)/2与根号n相近，通过爆破这个差值能够容易地计算出p+q，从而分解n  def detect4():      for i in range(21):          if i not in solved:              p\_q = iroot(n[i] , 2)[0]              for \_ in range(20000):                  p\_q += 1                  if iroot(p\_q\*\*2 - n[i],2)[1] == 1:                      tmp = iroot(p\_q\*\*2 - n[i],2)[0]                      p = (p\_q + tmp)                      q = (p\_q - tmp)                      if GetPlain(p , q , e[i] , c[i]):                          solved.append(i)  Pollard p-1 分解法  如果p-1为k-smooth，则p-1|k!，则通过gcd就能够将n分解，  def Pollard\_p\_1(N):      a = 2      f = a      # precompute      while 1:          for n in range(1,200000):              f = powmod(f, n, N)              if is\_prime(n):                  d = gcd(f-1, N)                  if 1 < d < N:                      return d , N//d                  elif d >= N:                      f = next\_prime(a)                      break          else:              break  def detect3():      for i in range(21):          if i not in solved:              tmp = Pollard\_p\_1(n[i])              if isinstance(tmp , tuple):                  p , q = tmp                  if GetPlain(p,q,e[i],c[i]):                      solved.append(i)  此外，还可以尝试因数碰撞法，即对每一个n计算公因数，若能够计算出非0公因数则能够直接将两个n进行分解。  def detect2():      for i in range(21):          for j in range(21):              if i != j:                  if 1 < gcd(n[i] , n[j]) < n[i]:                      p = gcd(n[i] , n[j])                      q1 = n[i] // p                      q2 = n[j] // p                      tmp1 = GetPlain(p , q1 , e[i],c[i])                      tmp2 = GetPlain(p , q2 , e[j],c[j])                      if tmp1 ==1:                          solved.append(i)                      if tmp2 == 1:                          solved.append(j)  用分解法能够分解1，2，6，10，18，19六个模数，可以直接计算私钥求解对应的明文  对于剩下的模数，用分解的办法是无法在短时间内分解的。因此需要寻找指数e的缺陷  共模攻击：  对于使用了相同的n相同m，不同e所对应的两个密文，我们可以通过共模攻击在不分解n的前提下求解出m  由裴蜀定理，对于互素的e有且仅有一对a1，a2，能够满足  则  def same\_module\_attack(N , e1 , e2 , c1 , c2):      d1 = invert(e1 , e2)      d2 = (d1 \* e1 - 1) // e2      true\_c2 = invert(c2 , N)      return (powmod(c1 , d1 , N) \* powmod(true\_c2 , d2 , N)) % N  def detect1():      for i in range(21):          for j in range(21):              if i != j and n[i] == n[j] and e[i] != e[j]:                  tmp = \_GetPlain(same\_module\_attack(n[i] , e[i],e[j],c[i],c[j]))                  if tmp == 1:                      solved.append(i)                      solved.append(j)  小指数广播攻击：  对于e=3与5的几个密文，如果他们所对应的明文相同，则可以通过小指数广播攻击来求解出m，用这种方法只求解出了e-=5的5个密文对应的明文，求不出e=3对应的明文，原因是e=3的三个明文不同，而e=5时五个明文相同  使用中国剩余定理，我们可以解出一个C = ci (mod ni)  而这个C = m^5 (mod n1n2...n5)，又因为m<ni ，所以C < n1n2...n5，所以该式在ZZ上成立，直接对C开5次方根即可求出m  def CRT(mi, ai):      M = reduce(lambda x, y: x \* y, mi)      ai\_ti\_Mi = [a \* (M // m) \* invert(M // m, m) for (m, a) in zip(mi, ai)]      return reduce(lambda x, y: x + y, ai\_ti\_Mi) % M  def small\_e\_boardcast\_attack(nlist , e , clist):      m = CRT(nlist , clist)      tmp = iroot(m , e)      if tmp[1] == 1:          return tmp[0]      else:          return 0  def detect5():      e = 5      num = [3,8,12,16,20]      nlist = [n[i] for i in num]      clist = [c[i] for i in num]      m = small\_e\_boardcast\_attack(nlist , e ,clist)      if \_GetPlain(m):          for i in num:              solved.append(i)  Coppersmith partical m方法：  Coppersmith在2004年提出了一种方法，将求解单元模方程的小解转化为求解格上最短向量问题，原理是对该方程的系数进行等价转化，让方程的系数绝对值尽可能小，让该方程的一个小解能够在整数范围内成立，则该方程能够通过二分法牛顿法等方法求解出根。而等价转化的这一步，又可以转化为几个向量的线性组合，以转化为格上的最短向量问题。而格上的最短向量问题，可以由格基规约算法LLL，BKZ或是由直接寻找svp的枚举法，筛法解决。  因此我们需要将rsa上的问题转化为一个求解模方程的小解的问题。当e=3时，虽然m本身较大，但是m的上半部分我们已知，未知仅有一个序号与8字节(64bit)的明文，序号可以直接爆破。而8字节的未知数是一个很小的数字，可以构造出一个以该8字节数为根的函数，通过调用coppersmith的求根法来解决问题。  设f = (app + m)^3 - c (mod n)，app为m的高位已知数，则我们可以通过coppersmith算法来求解出m  from Crypto.Util.number import \*  def GetPrefix(i):  res = '9876543210abcdef' + hex(i)[2:].rjust(8 , '0') + '0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000' + 16 \* '0'  return int(res , 16)  nlist = []  clist = []  e = 3  for j in range(3):  PR.<x> = PolynomialRing(Zmod(nlist[j]))  for i in range(21):  a = GetPrefix(i)  f = (a + x) ^ e - clist[j]  x0 = f.small\_roots(X = 2^64 , beta = 1)  if len(x0) != 0:  print(i ,long\_to\_bytes(int(x0[0])))  几次爆破后成功求解出三个对应的明文。  最后将其整合，实验结果如下 |
| **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）**  **没有问题** |
| **参考文献**  [1] Don Coppersmith: Finding a Small Root of a Univariate Modular Equation. EUROCRYPT 1996 (LNCS 1070, Springer): 155-165. |