In the puzzle, we can get the following information：

r1 =(A\*r0+B )%p

C0 = (m^e\*h^r0)%p

h^r0 = c0\*m^-e%p

C1 = (m\*h^r1)%p

=(m\*h^(A\*r0)\*h^(B))%p

=m\*(h^r0)^A\*h^(B)%p

C1\*h^(-B) = m\*(c0\*m^-e)^A%p

=m\*c0^A\*m^(-e\*A)%p

C1\*h^(-B)\*c0^(-A)=m^(1-e\*A)%p

In this way, the problem is simplified to an RSA problem

But e and φ(N) not co-prime.

Use some function force to find the n-th power of cipher.like AMM.

Then you can go through all the roots and find the correct one.

|  |
| --- |
| from Crypto.Util.number import \*  pk =(xxxxxxxxxxxxxxxxx)  c0,c1 = (xxxxxxxx,xxxxxxxxxx)  g, h, A, B, p, q = pk  R = IntegerModRing (p)  Rn = IntegerModRing (p-1)  k0 = c1 \* R(c0)^-1 \* (R(h)^Rn(A))^-1  k1 = Rn(B-1)^-1  y = R(c0)^Rn(B)\*R(h)^Rn(A)\*R(c1)^-1  ee = Rn(0xfaab\*B-1)  #factor ee in factordb.com  d0=Rn(factor(ee)[0])^-1  d1=Rn(factor(ee)[1])^-1  yy = (y^d0)^d1  def AMM(o, r, q):      import time      start = time.time()      print('\n----------------------------------------------------------------------------------')      print('Start to run Adleman-Manders-Miller Root Extraction Method')      print('Try to find one {:#x}th root of {} modulo {}'.format(r, o, q))      g = GF(q)      o = g(o)      p = g(random.randint(1, q))      while p ^ ((q-1) // r) == 1:          p = g(random.randint(1, q))      print('[+] Find p:{}'.format(p))      t = 0      s = q - 1      while s % r == 0:          t += 1          s = s // r      print('[+] Find s:{}, t:{}'.format(s, t))      k = 1      while (k \* s + 1) % r != 0:          k += 1      alp = (k \* s + 1) // r      print('[+] Find alp:{}'.format(alp))      a = p ^ (r\*\*(t-1) \* s)      b = o ^ (r\*alp - 1)      c = p ^ s      h = 1      for i in range(1, t):          d = b ^ (r^(t-1-i))          if d == 1:              j = 0          else:              print('[+] Calculating DLP...')              j = - discrete\_log(d, a)              print('[+] Finish DLP...')          b = b \* (c^r)^j          h = h \* c^j          c = c^r      result = o^alp \* h      end = time.time()      print("Finished in {} seconds.".format(end - start))      print('Find one solution: {}'.format(result))      return result  def findAllPRoot(p, e):      import time      import random      print("Start to find all the Primitive {:#x}th root of 1 modulo {}.".format(e, p))      start = time.time()      proot = set()      while len(proot) < e:          proot.add(pow(random.randint(2, p-1), (p-1)//e, p))      end = time.time()      print("Finished in {} seconds.".format(end - start))      return proot  mp = AMM(yy, ee, p)  p\_proot = findAllPRoot(p, ee)  def findAllSolutions(mp, proot, cp, p,e):      print("Start to find all the {:#x}th root of {} modulo {}.".format(e, cp, p))      start = time.time()      all\_mp = set()      for root in proot:          mp2 = (mp \* root) % p          if (pow(mp2, e, p) == cp):              all\_mp.add(mp2)      end = time.time()      print("Finished in {} seconds.".format(end - start))      return all\_mp  mps = findAllSolutions(mp, p\_proot, yy, p,ee)  from Crypto.Util.number import \*  for i in mps:      if("HIT" in long\_to\_bytes(i)):          print(long\_to\_bytes(i))  #HITCTF2021{Numb3r\_Th30ry\_1s\_Funny!} |