babyrsa

Category: Cryptography

you only need a little bit of meth for this...
flag format: isfcr{xxx}

Author: `Arrow#1334`

Flag: `isfcr{qu4dr4t1c_3quation5_f0r_th333_w1n}`

This challenge is based on the RSA Cryptosystem. You can read more about it [here](). There are many resources available online that teach you how RSA works. The attached wikipedia link is a great starting point.

The basic idea of RSA is as follows:

- There is some data/message `m` that needs to be privately sent through a public channel
- To do so, a "private key" and a "public key" are decided. The message is encrypted using the "public key" to derive a "ciphertext"
- The private key is only known to the sender
- The public key is known in the public channel to anyone
- The decryption of the ciphertext is only be possible if someone posseses the "private key"

The way RSA accomplishes this is by relying on the difficulty of factorisation of large numbers. We choose two prime numbers `p` and `q` and define the "modulus" `n = p * q`. We also choose an exponent `e`. Typically, `e` is chosen to be `65537` in most common implementations. The encryption is performed by calculating the following value `ciphertext = pow(m, e, n)` i.e. calculating `m` raised to the power of `e` and taking the result modulo `n`.

The "private key" comprises of `(p, q, e)`. The "public key" comprises of `(n, e)`. It is quite easy to derive the private key from the public key if `n` is small enough but for very large numbers, it is considered impossible to do in a reasonable amount of time. This is because even the best known integer factorisation techniques require O(n ** alpha) (alpha <= 0.5 approx.) time, which is very slow.

The decryption relies on knowing the private key. The original message can be recovered by first finding `phi = (p - 1) * (q - 1)` (the euler totient function of `n`), then finding `e = pow(e, -1, phi)` (the modular inverse of e under phi) and at last `m = pow(ciphertext, d, n)`. The mathematics behind all of this is better explained in the attached wikipedia link.

So, let's try solving the given challenge. The challenge has one flag divided into two parts. Both parts are encrypted with different keys. Ofcourse, without any additional knowledge, factoring `n1` and `n2` would be nearly impossible. But, we're lucky enough to know `p1 + q1` and `p2 - q2` in this implementation of RSA. Just by knowing those two values, we can easily find `p1`, `q1`, `p2` and `q2`. (It's similar to how you would solve a system of two variables given two equations)

We are given `n1 = p1 * q1` and that `k = p1 + q1`. We also know that `(p1 + q1) ** 2 + (p1 - q1) ** 2 == 4 * p * q`. We can substitue the corresponding values and solve for `p1 - q1`. Once we have that, we can easily figure out `p1` and `q1` by simple addition and subtraction. The procedure for `p2` and `q2` is exactly the same.

Here's the complete exploit script to retrive the flag:

```
import gmpy2
from Crypto.Util.number import long_to_bytes

e = 65537
n1 = 902116732427900015006283425659032470077750172023134716578847548335818317270520063719238207731823421862454543904946431221721186224579241879655449811353328624345211523371802763248155835871476640413996274623478357221567366037777900330448891041691600968642469109159908871740225407816627064033425495794659657796629
n2 = 1074018369963577768431675071234961167032483886803124535755357534031712132995892263342912901699516715676298968622876704796245797729061805755103510912598671852618982181974854023860261081335037928835649511651016574810652022705624911500605834601395517036645005351758118869293973584871563217648501074910629774 73251
c1 = 579394600109651619476553515339452786889145831120108747464753281253904457469973715220652454918667815153385840406296434559733298742093583404661236998425530460427318303507411917977696610692780498111381923820216212242612166999139252220983518475946187243968887526653752355586597763418509114678244800664551217233307
c2 = 752555745356220304568332745188918109169037289456760850218600423133611383664914638060517178255192650030256375594297486122092846806132364516492248308959682336316341691482731032607851766022583815814377931024158111263265995618802480302015064997435151308662578315265309949132147477042873996765248041477144283 2300
p1_plus_q1 = 194379609024044358335709577497241722417650862989860945172550344384753562256738656067284037962736215159523857711516418745267868500750379887270197529712270 54
p2_minus_q2 = 16113546084873111796097001182412369045900638132315676200686970420859492023010146845716692915253585371864529209514617657804527908574999409097105410895786 86

def attack_1 ():
  p1_minus_q1 = gmpy2.isqrt(p1_plus_q1 ** 2 - 4 * n1)
  p1 = (p1_plus_q1 + p1_minus_q1) // 2
  q1 = (p1_plus_q1 - p1_minus_q1) // 2
  assert(p1 * q1 == n1)

  phi = (p1 - 1) * (q1 - 1)
  d = pow(e, -1, phi)
  m = pow(c1, d, n1)

  return long_to_bytes(m).decode()

def attack_2 ():
  p2_plus_q2 = gmpy2.isqrt(p2_minus_q2 ** 2 + 4 * n2)
  p2 = (p2_plus_q2 + p2_minus_q2) // 2
  q2 = (p2_plus_q2 - p2_minus_q2) // 2
  assert(p2 * q2 == n2)

```

```
  phi = (p2 - 1) * (q2 - 1)
  d = pow(e, -1, phi)
  m = pow(c2, d, n2)

  return long_to_bytes(m).decode()

flag1 = attack_1()
flag2 = attack_2()
flag = flag1 + flag2

print(flag)
```