# Quantum

**Category**: “Van Halen Radiation Belt”

## Description

The challenge concerns the RSA cryptosystem, and it consists in breaking such scheme with the help of an oracle that gives away the result of the computation of the quantum part of Shor’s algorithm.

## Solution

The framework of the challenge is simply RSA, with access to some particular oracle. The server sends us a formatted public key, and then asks us for some value, we will call such value .

Before going into the challenge, we give a little bit of context. The RSA cryptosystem, and the mathematical assumptions it relies upon, are known to be broken in the quantum setting due to an algorithm of Peter Shor. This algorithm is split into two parts, the classical part and the quantum part. The quantum part of Shor consists in computing the period of the function : note that this is classically infeasible, assuming the factorization of to be unknown. The classical part of Shor uses the result of the quantum computation to obtain the order of in . From here, factorization of is (probabilistically) trivial. Assume . If is odd, one can obtain a nontrivial factor of by simply computing . If turns out to be even, we need to choose another value of , and ask the oracle for its order (hence the probabilistic nature of this algorithm), until we get an odd value of .

The oracle given to us in the challenge does exactly this, it “computes the quantum part” of Shor’s algorithm, and gives us the result. This makes for an easy factorization of , by simply following the steps of the classical component, as outlined above. Once is factored, we have an easy road ahead of us, we can decrypt all the information the server sends us concerning the AES-GCM instantiation with which the flag is encrypted, namely key, tag. At this point, we can all we need to decrypt the encryption of the flag locally.

### Quick considerations

It might be interesting to try and understand how the authors were able to set up an oracle “computing the quantum component of Shor”. In general, such computation is known to not be polynomial-time, hence infeasible for cryptographically-graded parameters. The question has a fairly trivial response, once we consider that the server the factorization of . By Lagrange’s Theorem, we know that the order of any element in a group divides the order of the group itself. This means that we can easily compute the order of a given element in , where , for known , by trying all possible divisors of . The smallest divisor of satisfying will be, by definition, the order of .

## Solution script

from pwn import remote  
from Crypto.PublicKey import RSA  
from base64 import b64decode  
from Crypto.Util.number import \*  
from random import randrange  
from math import gcd  
  
from Crypto.Cipher import PKCS1\_OAEP, AES  
  
host, port = 'quantum.quals2023-kah5Aiv9.satellitesabove.me', 5300  
  
ticket = b'ticket{juliet404343alpha4:GPcLzu\_hcmygpqc2y8YynpUU91FTobK561KtYl89IZy\_EpbJX\_VXxtuJVMa58klZ6A}'  
  
with remote(host, port) as chall:  
 chall.sendline(ticket)  
 chall.recvuntil(b'encrypted message we intercepted is: \n')  
 ctx = b''.join(chall.recvlines(9, False))  
 ctx = b64decode(ctx)  
  
 chall.recvuntil(b'The public key is:\n')  
 pemkey = b'\n'.join(chall.recvlines(9, False)) + b'\n'  
  
 key = RSA.import\_key(pemkey)  
 print(f'{key.n = }\n{key.e = }')  
  
 n, e = key.n, key.e  
  
 while True:  
 chall.sendline(str(n).encode())  
  
 a = randrange(1,n)  
  
 chall.sendline(str(a).encode())  
  
 chall.recvuntil(b'The quantum computer returned the following value for the period:\n')  
  
 r = int(chall.recvline().decode().strip())  
  
 if r%2:  
 continue  
  
 coso = pow(a, r//2, n)  
  
 if coso == n-1:  
 continue  
  
 p = gcd(coso+1, n)  
 q = gcd(coso-1, n)  
  
 assert p\*q == n and p != 1 and p != n  
  
 break  
  
 print(f'{p = }\n{q = }')  
  
 phi = (p-1)\*(q-1)  
  
 d = pow(e,-1,phi)  
  
 privkey = RSA.construct((n, e, d))  
  
 cipher = PKCS1\_OAEP.new(privkey)  
  
 enc = ctx[:256]  
 nonce = ctx[256:256+16]  
 mac = ctx[256+16:256+32]  
 ct = ctx[256+32:]  
  
 assert len(ct) == 112, f'{len(ct) = }'  
  
 symkey = cipher.decrypt(enc)  
  
  
 aes = AES.new(symkey, AES.MODE\_GCM, nonce=nonce)  
  
 pt = aes.decrypt\_and\_verify(ct, mac)  
  
 print(pt)  
 chall.interactive()