Tools used: SageMath

```
def test_encryption(self):
        plaintext = input('Plaintext: ').strip()

        m = int(plaintext, 16)
        cs = []

        while m:
                c = pow(m, self.e, self.n)
                cs.append(c)
                m//= self.n

        if len(cs) > 1:
                return 'Too many messages!'

        return 'Thanks for the message!'
```

Summary:

```
m<n  => Thanks for the message!
m>=n => Too many messages!
```

The binary search can take few mins:

```
from pwn import *

p = remote("134.122.105.9",31644)
reply=p.recvuntil(b'Option: ')

def getOne():
        p.sendline(b"1")
        reply = p.recvuntil(b'Option: ')
        print(reply)
        guess()


def guess():
        lower, upper = 10**616, 10**618
        actualKnown=1
        while (upper-lower!=1):
                knownNumbers=618-len(str(upper-lower))
                if(knownNumbers>actualKnown):
                        actualKnown=knownNumbers
                        print(str(upper)[:knownNumbers-2]+(618-knownNumbers)*"-")

                p.sendline(b"2")
                reply = p.recvuntil(b'Plaintext: ')
                p.sendline(bytes(hex((lower+upper)//2),"utf-8"))
                reply = p.recvuntil(b'Option: ')
                if("Too many messages" in str(reply)):
                        upper=(lower+upper)//2
                elif("Thanks for the message!" in str(reply)):
                        lower=(lower+upper)//2

        print("Done ! n=",upper)
```

```
getOne()
```

https://gchq.github.io/CyberChef/#recipe=From_Base(16)&input=MHg4Y2U3YTcyN2M0YTcwNDcwYWEzZDZiODcyZjgyZWYyNmM4ZmY1Yzc4MjBkNTc5MGFhNTNlOWRiZDFkOWYzMjhjODg0N2IyNjg4MTNjOWMxYmNhNTRjM2E4OTJjOWE4NDhlOTVmMzdiYjNjNDY3OTcxYWYzYTI5YTJlYTcwNmRkZTY2MmNhYTU5NTcyOGZmMDk0YjZjM2M2NmJkZGRjNjczM2Y0MjhjNWI4MGVmODFjMGRiZmE3ZjQ0MTlmMDhjYzZjZTdjZGUzMGRmMjAwNGZmODAzN2JhZjM2NDdjZjE4MTNjNTc3Y2ExMzAzZmI5MmYzMTk0MThlM2VkNGYzNmRjNDlkMzNlN2Q5MjQ3MWE1M2FlMmMwMjljZGZhMmI5MDM0YTZjYjhmM2I0NjhmOTE1NGE2NzU1YWZmMTNkOTIzYmQ3ZTZkNDlkMmU4ZGIzYjM0YjYxMzU2NzVkMWExMTIzNmU3Yzg2NDE3MTZjNTRmZTkxZGMyNjY3NzIwMDIzMmJhYWU4YTlkNTI5MzEwOWQ3MzMzNmYyMzlkOWE4OTA1YzdhMWI4MWFlYzU3ZDNkZjU1ZjMzMDJjMGNkZGJjZDc0MmVhMzAyYzExNTdmYTdiMTEzOGIzNmFjZDgyY2Q3MzE0Mml0ZmJmMjYwOTkxNTM2MTZjZDBkMjQ0ZGVhMmUxYw

Coppersmith's small public RSA exponent attack with partially known message

Since we know all the beginning of the message we can use this attack.

Then we implement this attack from https://latticehacks.cr.yp.to/rsa.html

```
class univariate_coppersmith:
    def __init__(self, f, N, X):
        self.f = f
        self.N = N
        self.X = X
        self.R = QQ['x']

    def gen_lattice(self,t=1,k=1):
        d = self.f.degree()
        dim = k*d+t
        A = matrix(IntegerRing(),dim,dim)
        x = self.R.0
        X = self.X

        monomial_list = [x^i for i in range(dim)]
        for i in range(k):
            for j in range(d):
                g = self.f(X*x)^i*(X*x)^j*self.N^(k-i)
                A[d*i+j] = [g.monomial_coefficient(mon) for mon in monomial_list]
        for j in range(t):
            g = self.f(X*x)^k*(X*x)^j
            A[k*d+j] = [g.monomial_coefficient(mon) for mon in monomial_list]

        weights = [X^i for i in range(dim)]
        def getf(M,i):
            return sum(self.R(b/w)*mon for b,mon,w in zip(M[i],monomial_list,weights))
        return A,getf

    def solve(self,t=1,k=1):
        A,getf = self.gen_lattice(t,k)
        B = A.LLL()
        roots = []
        for r,multiplicity in getf(B,0).roots():
            if mod(self.f(r),self.N) == 0:
                roots.append(r)
        return roots

def gen_random_coppersmith(a,rlen=150,d=3):
```

```
    N = 217...

    c = 177...

    x = ZZ['x'].0
    f = (x+a)^d-c
    X = 2^rlen
    return (f,N,X)

def test_coppersmith():
    for length in range(50):
        a = int(("Hey! This is my secret... it is secure because RSA is extremely strong and very
hard to break... Here you go: HTB{"+("0"*length)).encode("utf-8").hex(),16)
        (f,N,X) = gen_random_coppersmith(a)
        u = univariate_coppersmith(f,N,X)
        solutions=u.solve()
        if(len(solutions)>0):
            print(bytes.fromhex(hex(a+solutions[0])[2:]))

test_coppersmith()
```

We get the flag:

```
Hey! This is my secret... it is secure because RSA is extremely strong and very hard to break...
Here you go: HTB{*4*_*L*_*r*4*_*S*?*!}
```