COPPERSA LATTICE

There are plenty of bits of randomness in the key. This has to be secure!

I am so confident, that I am sharing a snippet of my implementation:

import random

base = random.getrandbits(2048)

p = next\_prime(base + random.getrandbits(256))

q = next\_prime(base + random.getrandbits(256))

n = p \* q

e = 65537

print("e = ", e)

print("n = ", n)

c = power\_mod(m, e, n)

print("c = ", c)

I think this was a basic RSA challenge but you should read about CopperSmith attack.

Well I automated this script using python and decrypt the RSA encryption. The script is below:

#!/usr/bin/python3

from Crypto.Util.number import inverse

e=65537

n=8281850967132278399574272688766937486036646313403007679588335903785669628431708760927341727806006769095252325575815709840401878674105658204057327337750902945521512357960818523078486774689928139816732080923197367563639383252762498921096166065153092144335239373370093976823925031794323976150363874930075228846801224430767428594024165529140949082062667410186456029480046489969338885725614510660737026927443934115006027747874368836800022473917576424175601374800697491622086825964475396316066082109998646438504664272000556702241576240616765962934452557847306066736505798267513078073147161755828577875047115978481485076227911405625234425533967247248864837854031634968764570599279385827285321806293661331225163255461894680337693227417748934924109356565823327149859245521513696171473417470936260563397398387972467438182651142096935931112668743912944902147582538985769095457203775208567489073198557073226907349118348902079942096374377432431441166710584381655348979330535397040250376989291669788189409825278457889980676574146044704329826483808929549888234303934178478274711686806257841293265249466735277673158607466360053037971774844824065612178793324128914371112619033111301900922374201703477207948412866443213080633623441392016518823291181

c=6407923537926201847312357068295079879508779752068254604904486842729636773279241546432035102141932853761974844472828552921133743850412718722424893044377874567625621282274625365299685502104113862870672461666586814138206797733946319875258776059721304226419810313489197076949529322847815009706727586961448584443159011118432142946962961532154723891985416387650240762711716865116844837968079333914181751979527853152286708153252001832721723040664452442266930832118353632114958540067674924812749763008217133300059446967170825813909142247660230309955433005706793802514554628379255160648976960069078223370104177403453404917998945232459801324103878906593528309460372271638119657797804398399482025063414403804134607772871958848100256643503372624214762343403925077455660522664025602043433142314759978192969519687720668535544914589329155338178120703060384042066182354031274600184116143293639032906542194564776766076911767759167772137229504115598174156646085123675283692418970988032320780636742598466655712520383055569607154074137271584433653335176877094399371749081016317705026349554938167377640856287458145646649292278971980553895419112860061073864077521131958519819285117031990498977039003918710661660868949818362940359852436185282868088342132

p=2877820523787450925749443223409676535526103461002156158445828224293671401993991478543020287097392331073352851877283061169200661485601861126177989029099279726556037706157577055235829106587295127330817726845862915284936240880512882371822435354503703582818585826639860414460347276612398615213473473435297607044419660432857760267782763215613805061134548930946450808892523918406477350229326879718460875896139320212120566207583969498664502640043503068067423704342779684140776844591504545219651693576024138161226306265777913216051021635043708034933142150577361241126706252795462441428424206966514776010058207246337809108763

q=2877820523787450925749443223409676535526103461002156158445828224293671401993991478543020287097392331073352851877283061169200661485601861126177989029099279726556037706157577055235829106587295127330817726845862915284936240880512882371822435354503703582818585826639860414460347276612398615213473473435297607044419660432857760267782763215613805061134548930946450808892523918406477350229326879718460875896139320212120566207583969498664502640043503068067423704342779684140776844591504545219651693576024138161226306265777913216051021635043708034966478921186010140653104995611058414645030602000549187384764408555794028217687

phi=(p-1)\*(q-1)

d=inverse(e, phi)

m=pow(c, d, n)

print(hex(m)[2:-1])

I used factordb to get p and q(factorise) from n. Here is the link : <http://www.factordb.com/>

This gave a hex string : 4354466c6561726e7b6e30745f746834745f7333637572335f34667433725f346c6c7

After converting this to ascii flag was clear : CTFlearn{n0t\_th4t\_s3cur3\_4ft3r\_4ll.}