The challenge looks fairly difficult for someone new to crypto. I'm kind of a noob in this area (not for long though hehe), but let's go through the steps I took anyway.

So… I looked at the files: private key part, public key and the secret.

First I looked up how to get the modulus of the public key, I ended up using openssl for this:

```
openssl rsa -pubin -in my.pub -modulus -text
```

I took note of the exponent `0x10001` and used python to strip the colons and get the modulus:

```
>>> print "00:ba:cd...".replace(" ", "").replace("\n", "").replace(":", "")
```

So I got the 1024-bit modulus, and since that's expressed in HEX form, I converted it into a decimal to get some ridiculous number. Again, I used python :P

```
>>> int("00bacd...", 16)
```

What's next is I went on `factordb.com` and pasted the decimal in there. The reason for that is, after reading up and understanding RSA (just about), I concluded that `n = p * q` where `n` is the modulus and `p` & `q` are the primes, and the primes are what were needed. I got a result:

```
10570796208942330718133134445050881347580442327501864726780301183323228790933543286750277085897914122428857455856744411456476812241092677201917577388557109
```

and

```
12409530116611113069722673907902939927077171750562461687594137012713380508978436132121375571419505917625282847052267327325475909667939575657231740096180389
```

were my primes. Nice, but where am I going with this?

Well, a little more reading and I found a neat little explanation of what needed to be done.

1. Get `phi(n) = (p-1) * (q-1)`
2. Find `d`, the modular multiplicative inverse of `e`: mod `phi(n)`

The key to encrypting and decrypting a message `m` is:

Encrypt: `c(m) = m ^ e mod n`

Decrypt: `m(c) = c ^ d mod n`

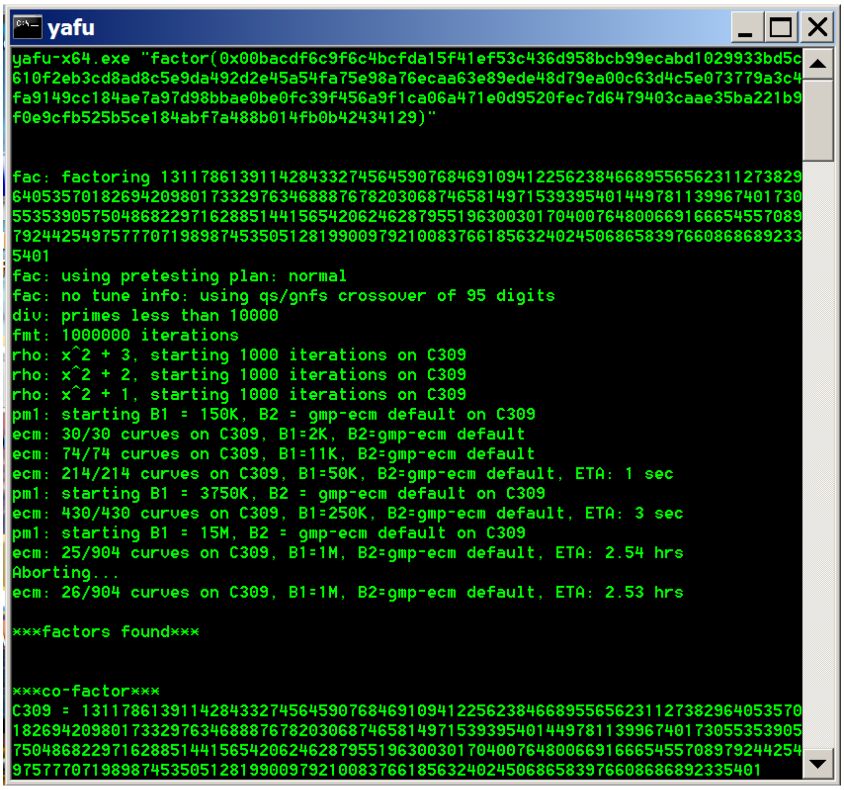
So find `phi(n)` and get its modular inverse, sounds pretty hard. Though there is an algorithm to help with the second part. Let's roll!

```
>>> e = 0x10001     # exponent
>>> n = 0x00bacd... # modulus
>>> p = 10570796... # 1st prime
>>> q = 12409530... # 2nd prime
>>> phi = (p-1) * (q-1)
>>> # defined functions found here ---> https://en.wikibooks.org/wiki/
Algorithm_Implementation/Mathematics/Extended_Euclidean_algorithm#Python
>>> d = modinv(e, phi) # used the functions and stored the output in d
>>> # used this answer ---> http://crypto.stackexchange.com/questions/25498/#25499 to
help me convert d into a PEM certificate. once done, openssl ftw
cat secret | openssl rsautl -decrypt -inkey awesome.key
{All H4il M3gatr0n! Def3at Otpimu$ -> W0rld Dom1nation!}
```

Hell yeah! I did it, and can't believe I didn't need the private key part. I thought it played a role. Anyhow, that's an ace up my sleeve I can surely brag about amirite :D

### ***What didn't work***

When it came to factorising the primes, I first tried to use a tool called Yafu – the result:




I ran this in a VM and the ETA didn't look pretty, neither did the sound of the fans I barely ever get to hear. So I hit ^C and aborted the process, later to realise that using an online tool was a better option, duh!

Big shoutout to:

1. blog.compass-security.com
2. yurichev.com
3. wikibooks.org
4. crypto.stackexchange.com
5. 0day.work

Gotta say this was pretty tough, I was new to this at the start and was sort of learning along the way. Great experience nevertheless, and certainly enjoyable :)