

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:

```
1057079620894233071813313444505088134758044232750186472678030118332322879093354328675027
7085897914122428857455856744411456476812241092677201917577388557109
```

and

```
1240953011661111306972267390790293992707717175056246168759413701271338050897843613212137
5571419505917625282847052267327325475909667939575657231740096180389
```

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 : $\text{mod } \phi(n)$

The key to encrypting and decrypting a message m is:

Encrypt: $c(m) = m^e \text{ mod } n$

Decrypt: $m(c) = c^d \text{ mod } n$

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 Domlnation!}
```

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:

```
yafu-x64.exe "Factor(0x00bacdf6c9f6c4bcfda15f41ef53c436d958bcb99ecabd1029933bd5c
610f2eb3cd8ad8c5e9da492d2e45a54fa75e98a76ecaa63e89ede48d79ea00c63d4c5e073779a3c4
fa9149cc184ae7a97d98bbae0be0fc39f456a9f1ca06a471e0d9520fec7d6479403caae35ba221b9
f0e9cfb525b5ce184abf7a488b014fb0b42434129)"

fac: factoring 13117861391142843327456459076846910941225623846689556562311273829
64053570182694209801733297634688876782030687465814971539395401449781139967401730
55353905750486822971628851441565420624628795519630030170400764800669166654557089
79244254975777071989874535051281990097921008376618563240245068658397660868689233
5401
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 95 digits
div: primes less than 10000
fmt: 1000000 iterations
rho: x^2 + 3, starting 1000 iterations on C309
rho: x^2 + 2, starting 1000 iterations on C309
rho: x^2 + 1, starting 1000 iterations on C309
pm1: starting B1 = 150K, B2 = gmp-ecm default on C309
ecm: 30/30 curves on C309, B1=2K, B2=gmp-ecm default
ecm: 74/74 curves on C309, B1=11K, B2=gmp-ecm default
ecm: 214/214 curves on C309, B1=50K, B2=gmp-ecm default, ETA: 1 sec
pm1: starting B1 = 3750K, B2 = gmp-ecm default on C309
ecm: 430/430 curves on C309, B1=250K, B2=gmp-ecm default, ETA: 3 sec
pm1: starting B1 = 15M, B2 = gmp-ecm default on C309
ecm: 25/904 curves on C309, B1=1M, B2=gmp-ecm default, ETA: 2.54 hrs
Aborting...
ecm: 26/904 curves on C309, B1=1M, B2=gmp-ecm default, ETA: 2.53 hrs

***factors found***

****co-factor****
C309 = 1311786139114284332745645907684691094122562384668955656231127382964053570
18269420980173329763468887678203068746581497153939540144978113996740173055353905
75048682297162885144156542062462879551963003017040076480066916665455708979244254
9757770719898745350512819900979210083766185632402450686583976608686892335401
```

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 :)