

Elements (RSA Crypto) Solution Write Up Zerodays 2018 Dublin

By Francis Long

The RSA encryption algorithm is a cornerstone of modern secure communications. It is used in securing web pages (https), digital signatures and even satellite tv transmissions. It is an asymmetric algorithm, that is to say that it uses two encryption keys, one private and one public. If one key is used to encrypt then the other key is used to decrypt.

In the CTF challenge we are supplied with three variables used for the RSA process. We have the modulus (n), the exponent (e) and one of the two essential prime numbers (p). All these values are in hexadecimal comma separated format. Although the private key is the only key required to decrypt the flag, both keys can be calculated from the given values.

Step One: Find the Public Key (optional)

The following reference was used,

<https://stackoverflow.com/questions/11541192/creating-a-rsa-public-key-from-its-modulus-and-exponent>

A file called 'def.asn1' is required, using the format below, note the section in red.

```
# Start with a SEQUENCE
asn1=SEQUENCE:pubkeyinfo
# pubkeyinfo contains an algorithm identifier and the public key wrapped
# in a BIT STRING
[pubkeyinfo]
algorithm=SEQUENCE:rsa_alg
pubkey=BITWRAP,SEQUENCE:rsapubkey

# algorithm ID for RSA is just an OID and a NULL
[rsa_alg]
algorithm=OID:rsaEncryption
parameter=NULL

# Actual public key: modulus and exponent
[rsapubkey]
```

n=INTEGER:0x%%MODULUS%%
e=INTEGER:0x%%EXPONENT%%

The values for the modulus (n) and the exponent are substituted in integer form. The following online resource was used to strip the colons from the hexadecimal values.

<http://string-functions.com/hex-string.aspx>

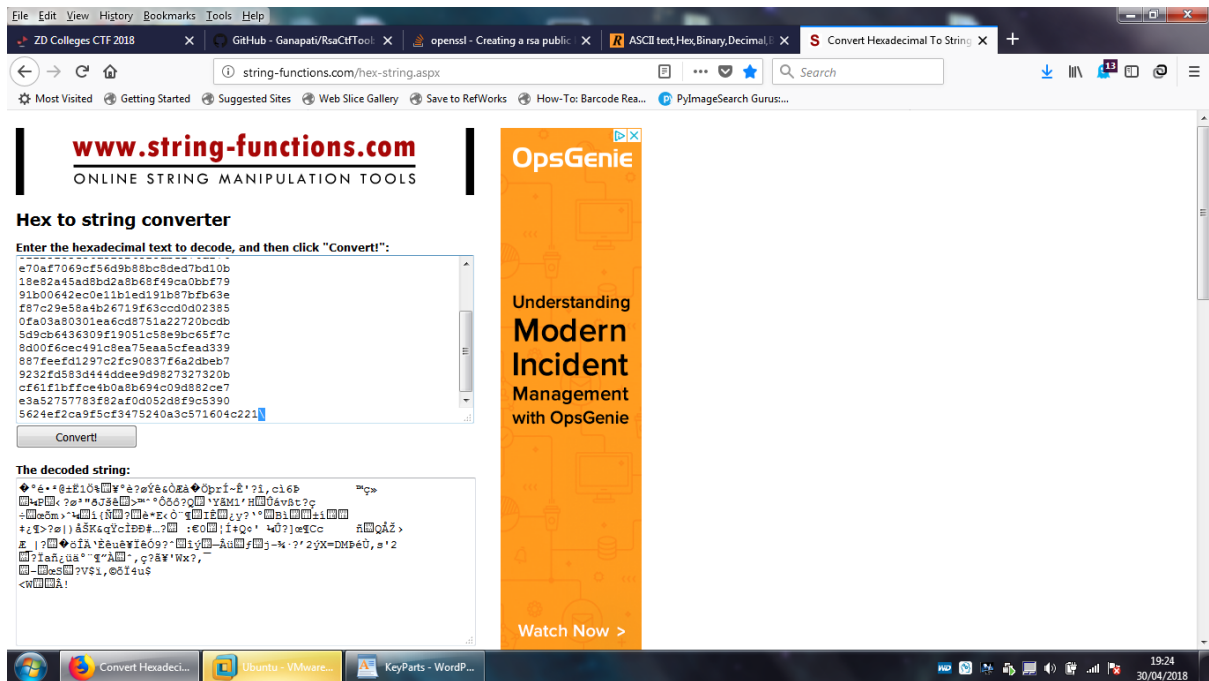


Fig 1: Stripping the colons from the hexadecimal numbers

Then the numbers can be converted to integers (decimal) using this website.

<https://www.rapidtables.com/convert/number/ascii-hex-bin-dec-converter.html>

```
def.asn1 (~/Desktop/Zerodays2018/RsaCtfTool-master) - gedit
1 # Start with a SEQUENCE
2 asn1=SEQUENCE:pubkeyinfo
3
4 # pubkeyinfo contains an algorithm identifier and the public key wrapped
5 # in a BIT STRING
6 [pubkeyinfo]
7 algorithm=SEQUENCE:rsa_alg
8 pubkey=BITWRAP,SEQUENCE:rsapubkey
9
10 # algorithm ID for RSA is just an OID and a NULL
11 [rsa_alg]
12 algorithm=OID:rsaEncryption
13 parameter=NULL
14
15 # Actual public key: modulus and exponent
16 [rsapubkey]
17
18 n=INTEGER:0x0192711941552462475913612820113378219196115151142105713212441961351821101721
19
20 e=INTEGER:0x0176233149178641772034921437201651762322482212343821019822402142541142051262
21
22
```

Fig 2: def.asn1 file with modulus (n) and exponent (e) substituted.

The next step is to create the public key using openssl.

```
ubuntu@ubuntu:~/Desktop/Zerodays2018/RsaCtfTool-master$ openssl asn1parse -genconf def.asn1 -out pubkey.der -noout
ubuntu@ubuntu:~/Desktop/Zerodays2018/RsaCtfTool-master$ openssl rsa -in pubkey.der -inform der -pubin -out pubkey.pem
writing RSA key
ubuntu@ubuntu:~/Desktop/Zerodays2018/RsaCtfTool-master$ openssl rsautl -verify
no keyfile specified
unable to load Private Key
ubuntu@ubuntu:~/Desktop/Zerodays2018/RsaCtfTool-master$ sudo cat pubkey.pem
[sudo] password for ubuntu:
-----BEGIN PUBLIC KEY-----
MIIcVjANBgkqhkiG9w0BAQEFAAOCAQsAMIICpgKCAU4ZJxGUFVJGJHWRNhKcARM3
ghkZYRUVEUIQVxMhJEGWE1GCEQFyEQM2YYh4IRg4gSAgglEZEGMkcggQIZASJRUD
gSBFFoQoIUYSAYkYRkYEkQpcRBhWQYJ0IIHhJIIIRIMUMjmJEUEBQxdxACSAS
GSKRGCMsVCGBJxJ0ISFxyJCIEgQYhAVULCUJBESIGehNVIEUhbZCSGSaBGBOSAG
YiYwGRZnFhd2UUQ1eBNBWCfILyERh4hBFSORgWERCZEXESISMSJRmRlyFigUh0Eg
QReGIG0IglRAXYBMikTIIkRNWlpMTgwETNjEhQQEHiiYkczAKYQMRh2FUNRJRIh
JBYIACCCSSYUHuWEIcWkTEQUSIWF0IjEYVlJWETWTFIUSIjc0JIFDEhFEI3dxGC
WUJBIGeMQS04SAFRdpIxdIERIFWXLxZ5kiQVERIDAOIBUBdImxSReGQXcgNJIUNY
AWUXYjIkgIeJ04IQGYIkahQlQRQgUSYgISI4RjKjZUIkVMjEYcVGIGBjXoSSBeT
QkB0FUI0FDYhUXNhdIeIRSRIEYFFISJ3dJFGcIehkiURgiMRYjEQJHYVYkUQKVUT
YvgUEjCSMgkRjCMkjpFzESIQFoGCFDcyAhEZESEUUXZmYjYUFxdyNyUnE1GRGCVI
SBjEEIkTh10BExWZkg0ggg1EzFRYFgShIMBZIBRNyEWISmhiCGZHVYJnmZJBFE
gRLxQhVRmJUSQUECRiBhIhRSACNBfyNBZSByNCEVctYScjgLMYFRGUJSFEExEnEG
RRkBgxRLAIOIYWh3IiIzIXEwEVOVARiHlyQRkSUIKBdhaBghSBkhVxNkQjEicWU5
hxIGMTAXUTVFFDFWgxRIY2ISRBaSR5B1IRc2EGCHIKGUMw==
-----END PUBLIC KEY-----
```

Fig 3: Creating the public key.

The first command creates an intermediate der file. The two openssl commands necessary are

```
openssl asn1parse -genconf def.asn1 -out pubkey.der -noout
```

```
openssl rsa -in pubkey.der -inform der -pubin -out pubkey.pem
```

The product is the public key pubkey.pem.

The public key can be tested and verified using the command

```
openssl rsa -pubin -inform PEM -text -noout <
pubkey.pem
```

```
ubuntu@ubuntu:~/Desktop/Zerodays2018/RsaCtfTool-master$ openssl rsa -pubin -inform PEM -text -noout < pubkey.pem
Public-Key: (2669 bit)
Modulus:
 19:27:11:94:15:52:46:24:75:91:36:12:82:01:13:
 37:82:19:19:61:15:15:11:42:10:57:13:21:24:41:
 96:13:51:82:11:01:72:11:03:36:61:88:78:21:18:
 38:81:20:20:82:21:19:12:03:24:72:08:10:21:90:
 12:25:15:03:81:20:45:16:84:28:21:46:12:01:89:
 16:61:19:18:12:44:29:71:10:61:59:64:18:27:42:
 08:16:12:48:20:14:48:31:43:23:98:91:14:10:14:
 31:77:10:02:35:22:92:19:22:91:18:23:12:54:21:
 81:27:12:50:89:21:71:43:22:42:20:48:10:62:10:
 15:52:50:94:24:11:12:22:01:21:35:52:04:52:10:
 73:09:21:92:68:11:81:39:20:06:62:26:16:81:16:
 67:16:17:76:51:44:35:78:13:41:58:21:48:97:21:
 11:87:88:41:15:23:91:81:61:11:71:91:17:11:22:
 12:31:22:51:99:19:72:16:28:14:87:41:20:41:17:
 86:22:01:74:20:69:51:03:16:01:32:29:13:22:29:
 11:35:69:69:31:38:30:11:33:63:12:14:10:12:12:
 22:62:47:19:02:46:10:31:18:76:15:43:51:25:12:
 21:24:16:22:20:20:02:09:24:98:52:15:16:12:20:
 96:91:31:10:53:92:16:17:42:23:11:85:62:25:61:
 13:59:31:62:51:22:23:73:42:48:14:31:21:14:42:
 37:77:11:82:59:42:41:20:61:26:41:24:38:48:01:
 51:76:92:31:76:21:11:88:55:97:97:16:79:92:24:
 15:11:12:03
Exponent:
 17:62:33:14:91:78:64:17:72:03:49:21:43:72:01:
 65:17:62:32:24:82:21:23:43:82:10:19:82:24:02:
 14:25:41:14:20:51:26:20:23:92:38:44:99:23:65:
 42:22:91:53:23:11:87:15:18:88:01:27:13:92:48:
 17:93:42:40:74:15:42:34:14:36:21:53:13:61:76:
 21:22:45:24:48:11:81:45:89:22:77:74:91:46:72:
 21:21:92:25:11:82:23:11:62:31:16:24:76:15:62:
```

Fig 4: Testing the public key.

One can even interrogate the public key to test the strength of the encryption (degree of entropy). The following command is used.

```
openssl rsa -in pubkey.pem -pubin -text -modulus
```

```
ubuntu@ubuntu:~/Desktop/Zerodays2018/RsaCtfTool-master$ openssl rsa -in pubkey.pem -pubin -text -modulus
Public-Key: (2669 bit)
Modulus:
 19:27:11:94:15:52:46:24:75:91:36:12:82:01:13:
 37:82:19:19:61:15:15:11:42:10:57:13:21:24:41:
 96:13:51:82:11:01:72:11:03:36:61:88:78:21:18:
 38:81:20:20:82:21:19:12:03:24:72:08:10:21:90:
 12:25:15:03:81:20:45:16:84:28:21:46:12:01:89:
 16:61:19:18:12:44:29:71:10:61:59:64:18:27:42:
 08:16:12:48:20:14:48:31:43:23:98:91:14:10:14:
 31:77:10:02:35:22:92:19:22:91:18:23:12:54:21:
 81:27:12:50:89:21:71:43:22:42:20:48:10:62:10:
 15:52:50:94:24:11:12:22:01:21:35:52:04:52:10:
 73:09:21:92:68:11:81:39:20:06:62:26:16:81:16:
 67:16:17:76:51:44:35:78:13:41:58:21:48:97:21:
 11:87:88:41:15:23:91:81:61:11:71:91:17:11:22:
 12:31:22:51:99:19:72:16:28:14:87:41:20:41:17:
 86:22:01:74:20:69:51:03:16:01:32:29:13:22:29:
 11:35:69:69:31:38:30:11:33:63:12:14:10:12:12:
 22:62:47:19:02:46:10:31:18:76:15:43:51:25:12:
 21:24:16:22:20:20:02:09:24:98:52:15:16:12:20:
 96:91:31:10:53:92:16:17:42:23:11:85:62:25:61:
 13:59:31:62:51:22:23:73:42:48:14:31:21:14:42:
 37:77:11:82:59:42:41:20:61:26:41:24:38:48:01:
 51:76:92:31:76:21:11:88:55:97:97:16:79:92:24:
 15:11:12:03
Exponent:
 17:62:33:14:91:78:64:17:72:03:49:21:43:72:01:
 65:17:62:32:24:82:21:23:43:82:10:19:82:24:02:
 14:25:41:14:20:51:26:20:23:92:38:44:99:23:65:
 42:22:91:53:23:11:87:15:18:88:01:27:13:92:48:
 17:93:42:40:74:15:42:34:14:36:21:53:13:61:76:
 21:22:45:24:48:11:81:45:89:22:77:74:91:46:72:
 21:21:92:25:11:82:23:11:62:31:16:24:76:15:62:
 45:10:91:55:13:61:88:14:12:37:12:32:09:11:24:
 23:24:26:91:73:13:92:10:16:81:82:14:37:32:02:
 11:19:11:21:14:51:76:66:62:36:14:17:17:72:37:
 25:27:13:51:91:18:26:22:48:12:44:12:29:13:87:
 53:81:13:15:99:92:04:20:82:08:35:13:31:51:60:
```

Fig 5: Using Openssl to output the modulus (decimal) and test the key strength. (2669 bit)

Step 2: Find the Private Key

Without going into too much mathematics of the RSA algorithm it is useful to remember that the modulus (n) is the product of two random prime numbers called p and q . (If these two numbers are not random and can be predicted, then the RSA algorithm can be attacked and defeated. Recently in Estonia 250,000 public identity cards were compromised because of this. The attack is known as a Coppersmith attack). The values n and p are supplied in the keyparts file.

Let $n = p \times q$, therefore $n/p = q$

These are huge numbers so calculators and online sites are not enough to accurately process them. Fortunately python comes with the capacity handle large integers, these are known as long integers or longs. (You know you are famous when you have a type of integer named after you).

This is what the script looks like.



```
KeyParts.py (~/Desktop/Zerodays2018/RsaCtfTool-master) - gedit
1 #!/usr/bin/python
2
3 import sys
4 import os
5
6 n = int
7 ("0x00c047c29bf6f73b8880c9854edbc473978e6907847c04c487b60b00ac6e2142bc4ed3535878d0ddbfcbf7d066be0cfb3226782da82a529278bda677b5f41d476
8 16)
9
10 e =
11 ("0x00b0e995b240b1cb31d62514a5b0e8f8ddea26d2c6e000d6fe72cd7eca27ee2c63ec36de0999e7bb0fbc507f8bf8b322f04a9aea8f3e9988b0d4f5f451129159e3
12 16)
13 p = int
14 ("0x00e544bf7823df3b55bbb0674dae0883e1c6f94bf40a5cbf729ebb4e851df990a87bfa2cacda94c84494997f4cccebb596c290f696bb088dfe3fcef341853bc78
15 16)
16
17 print "The value of q is " + str(n // p) + "\n"
18 print "The value of p is " + str(p)
19
```

Fig 4: Python script (KeyParts.py) to derive q , the prime number companion to p .

The important part is that python translates the hexadecimal (base 16) value of n and p into a long integer by wrapping it.

```
n = int("0x00c047029", 16)
```

```

ubuntu@ubuntu:~/Desktop/Zerodays2018/RsaCtfTool-master$ python KeyParts.py
The value of q is 15076686455371163858312117700578756207621279430827706875658784550379911616484185258600193066165273795847127912
42718358475307074461772344131868422584678345548091062250764121268079167840596629752120946726649897341942151078168313279946935742
24205038627660745712989844949617859890199429390852542189116777245203697

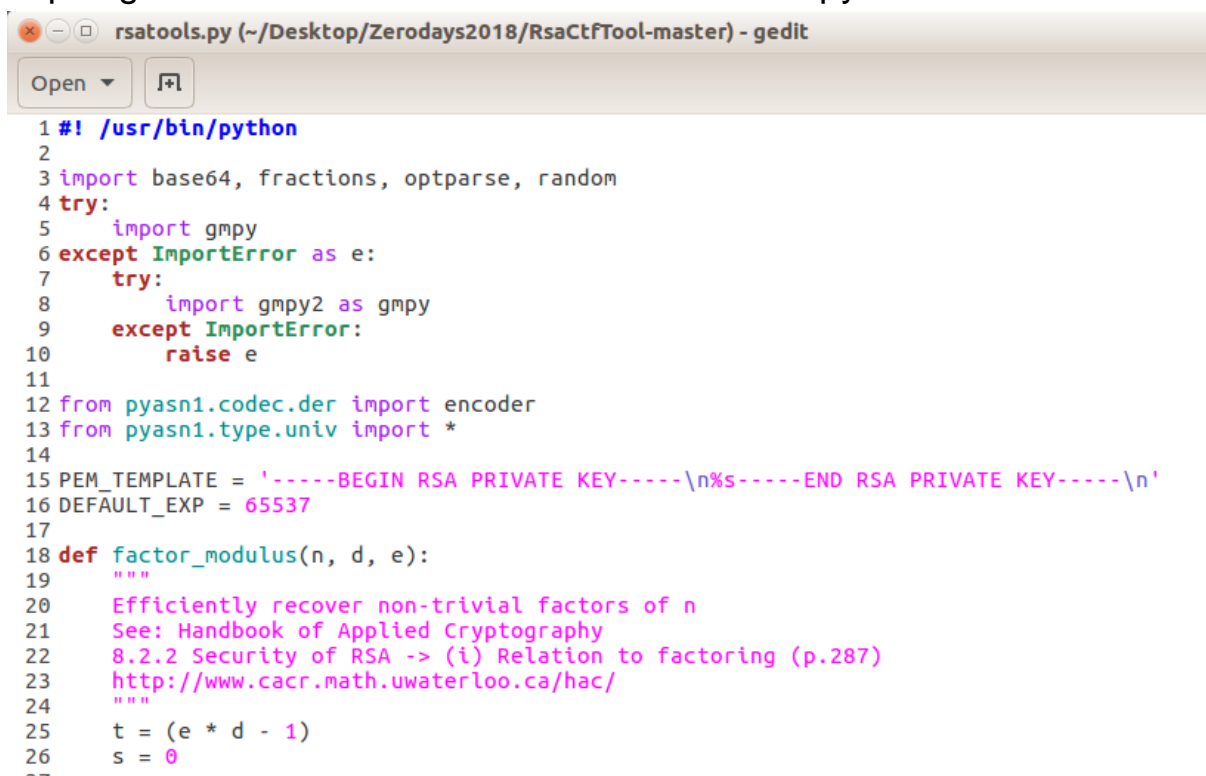
The value of p is 16099784853282095458374552791655903301268757324098695701802795217064755627822973985310681479028764382284737338
97835305646886544330274888321654451494306165957132001007645653687171099306576795654532113510170948529815610402640088182170619016
25853341051335720226950814944807522850075798109076779405024218258908283

```

Fig 5: Output of calculation to derive q.(KeyParts.py)

Once the two prime numbers p and q are known then it is possible to build the private key. The following python code was discovered on the website url, written by Sylvian Pelissier.

<https://github.com/ius/rsatool/blob/master/rsatool.py>



```

rsatools.py (~/Desktop/Zerodays2018/RsaCtfTool-master) - gedit
Open  [icon]

1 #! /usr/bin/python
2
3 import base64, fractions, optparse, random
4 try:
5     import gmpy
6 except ImportError as e:
7     try:
8         import gmpy2 as gmpy
9     except ImportError:
10         raise e
11
12 from pyasn1.codec.der import encoder
13 from pyasn1.type.univ import *
14
15 PEM_TEMPLATE = '-----BEGIN RSA PRIVATE KEY-----\n%s-----END RSA PRIVATE KEY-----\n'
16 DEFAULT_EXP = 65537
17
18 def factor_modulus(n, d, e):
19     """
20     Efficiently recover non-trivial factors of n
21     See: Handbook of Applied Cryptography
22     8.2.2 Security of RSA -> (i) Relation to factoring (p.287)
23     http://www.cacr.math.uwaterloo.ca/hac/
24     """
25     t = (e * d - 1)
26     s = 0

```

Fig 6: Sample code for rsatool.py, used to create the private key from two prime numbers p and q.

Here is the linux command in total.

```

ubuntu@ubuntu:~/Desktop/Zerodays2018/RsaCtfTool-
master$ python rsatools.py -p
16099784853282095458374552791655903301268757324098695
70180279521706475562782297398531068147902876438228473
73389783530564688654433027488832165445149430616595713

```

```
20010076456536871710993065767956545321135101709485298
15610402640088182170619016258533410513357202269508149
44807522850075798109076779405024218258908283 -q
15076686455371163858312117700578756207621279430827706
87565878455037991161648418525860019306616527379584712
79124271835847530707446177234413186842258467834554809
10622507641212680791678405966297521209467266498973419
42151078168313279946935742242050386276607457129898449
49617859890199429390852542189116777245203697 -o
private.pem
```

Here is the output,

Using (p, q) to initialise RSA instance

```
n=c047c29bf6f73b8880c9854edbc473978e6907847c04c487b60b00ac6
e2142bc4ed3535878d0ddbfcfb7d066be0cfb3226782da82a529278bda6
77b5f41d476a9f40b64ad0a1f8c92c538fef5972652bb164ebe5dbe576e7f
eda7f0c3259d98fe0dc306ad29bfa5e180b7ac9d537cc346b1e5cc044768
bc842e2a8744310b14190234e869e02946115764e08290fefb53d75bfab
7a0c1fe1c7c5d81c0e5729cc7556dcaece5f67a0841d84e57138605d8a1
e71241fd665d4e2f7bef667764c9a237d7a7ca2dcc8d1f955d7a102d1458
36927d8aedef7638e13d875da2057aed22f88f7990ed4d76195ef1ce7e29f
354500fb00917b0d3bc376161a763e0970bcb
```

e = 65537 (0x10001)

```
d=b0e995b240b1cb31d62514a5b0e8f8ddea26d2c6e000d6fe72cd7eca2
7ee2c63ec36de0999e7bb0fbc507f8bf8b322f04a9aea8f3e9988b0d4f5f4
51129159e34d31924815dbe176df74e70af7069cf56d9b88bc8ded7bd10
b18e82a45ad8bd2a8b68f49ca0bbf7991b00642ec0e11b1ed191b87bfb6
3ef87c29e58a4b26719f63ccd0d023850fa03a80301ea6cd8751a22720b
cdb5d9cb6436309f19051c58e9bc65f7c8d00f6cec491c8ea75eaa5cfead
339887feefd1297c2fc90837f6a2dbeb79232fd583d444dde9d98273273
20bcf61f1bffce4b0a8b694c09d882ce7e3a52757783f82af0d052d8f9c53
905624ef2ca9f5cf3475240a3c571604c221
```

```
p=e544bf7823df3b55bbb0674dae0883e1c6f94bf40a5cbf729ebb4e851df
990a87bfa2cacda94c84494997f4cccebb596c290f696bb088dfe3fcef341
853bc7893798ac6d7c1dffeb228808bec247d731d1be26078e7a6d55517
```


eb79185d52fd7ee8bfe195d538667f71db2e130f28069587e910de73d0f4
4a9c6f8cb608ca47b

q=d6b2faf4d532fcd7e72e4419c23672bcadeefab7203b8bce8f8d4d5044
5da87d9cc36928ac6c1b02bfa301fda3919c6fdf079b97906401bc2221b1
0d17a7864e1bb265f9507b5d0cfc01a48bb8061e6ad914a2c88666efa81
2805722a59fd844d12dc4833c2c9d8074c37e22a7e002e9ba9014f06046
14f970a27aba18fc5cf1

Saving PEM as private.pem

Now that the private key has been derived, this is all that is required to decrypt the flag.enc file, which is also supplied (Step 3).

Step 3: Decrypt the flag using openssl

```
ubuntu@ubuntu:~/Desktop/Zerodays2018/RsaCtfTool-master$ openssl rsautl -decrypt -in flag.enc -out  
message.txt -inkey private.pem
```

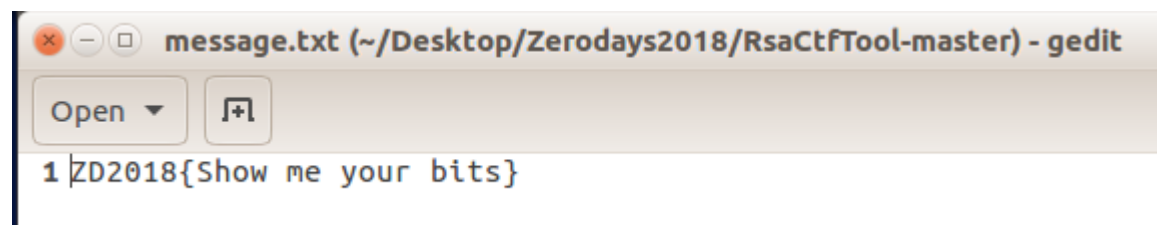


Fig 7: Decrypting the flag.

*Please do not re-use images from this report.