RSA

Question 1. Create a 2048bit RSA key-pair using openssl. Write your full name in a text file and encrypt it with your private key. Using OpenSSL extract the public modulus and the exponent from the public key. Publish your public key and the encrypted text on your wiki page

To Generate a 2048 bit RSA key:

```
kalachkar@desktop-15:~/Desktop$ openssl genrsa -des3 -out private.pem 2048
Generating RSA private key, 2048 bit long modulus
....+++
.....+++
unable to write 'random state'
e is 65537 (0x10001)
Enter pass phrase for private.pem:
Verifying - Enter pass phrase for private.pem:
```

Now, we encrypt the file myName.txt:

```
kalachkar@desktop-15:~/Desktop$ openssl rsautl -encrypt -inkey public.pem -
pubin -in myName.txt -out encryptedMyName.txt
```

If you look at the file it’s just binary junk, nothing very useful to anyone. that's why I will upload the encrypted file it self

The Encrypted File : encryptedmyname.txt.

File Contents:

```
S□□ÞÎ□iuZÂ□□□□□*«€uxÿç□øÌ1ÆZ□æ7îTžmK"Í)¢□Ë□Â□qÞ□□$□□□□öò[šþ
□□□Ò□«□ò□ç%³ù□æÁ¹^ô;íiX□J□□□G□n%□□.é□"-□F□-
š`¬Ü#V□óy{□□\□g□□E□]£]□ŠP□$□□`*Ø□=□Øs□¡I=o1
G□r□3ÞŸÕ6T□□q□ŸžÜ□□W□¹Ð□□y□fTÇWÂ□Z·ã□>.Y□`MÚŸ□-
ïdX¡6 õÔWó□□õ>¯£□ŠNÑ□□r^é4³N~OP□õØÇ¬DUø©³□âP□À□µDoQO□ÍRºwàÛo
```

Now, We need to export the RSA Public Key to a File: I will follow a great instructions on stackoverflow and I put the instruction URL below

```
kalachkar@desktop-15:~/Desktop$ openssl rsa -in private.pem -outform PEM -
pubout -out public.pem

Enter pass phrase for private.pem:
writing RSA key
```

My public key:

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvpqQ++xu+Ubo3ydsBuPb
hgScNvhEEwfT63iFMvwgF2RltSZZUDAwZC+iqj9BWJRUZSuknUyAPGBjlINDWMxl
Z9LQtfOtyaGcH0sOaBm3k7gfFBA72/HZlFbvc7IDhkfvv4RoJST/K3UJWyXPgvU3
```

```
F8Rin7LIF2pVFO4pYkqdR2UqDMTVGO3lzdAXWAGWB7h914w9Wf/1p54vag1OhDRJ
DELETED
-----END PUBLIC KEY-----
```

Using OpenSSL extract the public modulus and the exponent from the public key

```
kalachkar@desktop-15:~/Desktop$ PUBKEY=`grep -v -- ----- public.pem | tr -d
'\n'`
```

Then we can look at the ASN.1 Structure:

```
kalachkar@desktop-15:~/Desktop$ echo $PUBKEY | base64 -d | openssl asn1parse
-inform DER -i

    0:d=0  hl=4 l= 290 cons: SEQUENCE
    4:d=1  hl=2 l=  13 cons:  SEQUENCE
    6:d=2  hl=2 l=   9 prim:   OBJECT            :rsaEncryption
   17:d=2  hl=2 l=   0 prim:   NULL
   19:d=1  hl=4 l= 271 prim:  BIT STRING
```

The modulus and public exponent are in the last BIT STRING, offset 19, so use -strparse:

```
kalachkar@desktop-15:~/Desktop$ echo $PUBKEY | base64 -d | openssl asn1parse
-inform DER -i -strparse 19
```

It will give the following output:

```
    0:d=0  hl=4 l= 266 cons: SEQUENCE
    4:d=1  hl=4 l= 257 prim:  INTEGER
:BE9A90FBEC6EF946E8DF276C06E3DB86049C36F8441307D3EB788532FC20176465B52659503
030642FA2AA3F41589454652BA49D4C803C606394834358CC6567D2D0B5F3ADC9A19C1F4B0E6
819B793B81F14103BDBF1D99456EF73B2038647EFBF84682524FF2B75095B25CF82F53717C46
29FB2C8176A5514EE29624A9D47652A0CC4D518EDE5CDD01758019607B87DD78C3D59FFF5A79
E2F6A0D4E8434499C98E69DC01B5DCE977F27F89A163B76544D1BEFC51CDE6FDE786939081CC
28FFA32B8F9583461974C930B91F93B18DB13AC2C4784F35FFFAE7D9B97FC6D5612B9470DE17
3072E4A7305954C774FAC16FDEA2A323A3498240E9B7B6726D84B182D
  265:d=1  hl=2 l=   3 prim:  INTEGER           :010001
```

These two integers are the modulus and the public exponent in hexadecimal.

Another Easy way to do that directly (in case we want to put into a script):

```
kalachkar@desktop-15:~/Desktop$ openssl rsa -pubin -inform PEM -text -noout
< public.pem

Public-Key: (2048 bit)
Modulus:
    00:be:9a:90:fb:ec:6e:f9:46:e8:df:27:6c:06:e3:
    db:86:04:9c:36:f8:44:13:07:d3:eb:78:85:32:fc:
    20:17:64:65:b5:26:59:50:30:30:64:2f:a2:aa:3f:
```

```
    41:58:94:54:65:2b:a4:9d:4c:80:3c:60:63:94:83:
    43:58:cc:65:67:d2:d0:b5:f3:ad:c9:a1:9c:1f:4b:
    0e:68:19:b7:93:b8:1f:14:10:3b:db:f1:d9:94:56:
    ef:73:b2:03:86:47:ef:bf:84:68:25:24:ff:2b:75:
    09:5b:25:cf:82:f5:37:17:c4:62:9f:b2:c8:17:6a:
    55:14:ee:29:62:4a:9d:47:65:2a:0c:c4:d5:18:ed:
    e5:cd:d0:17:58:01:96:07:b8:7d:d7:8c:3d:59:ff:
    f5:a7:9e:2f:6a:0d:4e:84:34:49:9c:98:e6:9d:c0:
    1b:5d:ce:97:7f:27:f8:9a:16:3b:76:54:4d:1b:ef:
    c5:1c:de:6f:de:78:69:39:08:1c:c2:8f:fa:32:b8:
    f9:58:34:61:97:4c:93:0b:91:f9:3b:18:db:13:ac:
    2c:47:84:f3:5f:ff:ae:7d:9b:97:fc:6d:56:12:b9:
    47:0d:e1:73:07:2e:4a:73:05:95:4c:77:4f:ac:16:
    fd:ea:2a:32:3a:34:98:24:0e:9b:7b:67:26:d8:4b:
    18:2d
Exponent: 65537 (0x10001)
```

Sources:

1- https://rietta.com/blog/2012/01/27/openssl-generating-rsa-key-from-command/

2- https://www.devco.net/archives/2006/02/13/public_-_private_key_encryption_using_openssl.php

3- https://stackoverflow.com/questions/3116907/rsa-get-exponent-and-modulus-given-a-public-key

Question 2. Assuming that you are generating a 1024bit RSA key and the prime factors have a 512bit length, what is the probability of picking the same prime factor twice ? Explain your answer. Hint: How many primes with length 512bit or less exist?

To solve this problem, First, lets count how many primes with length 512-bit or less:

The prime number theorem states that π(x), or the number of primes less than or equal to x obeys the following property:

Meaning that we can approximate the number of primes less than x by x/ ln x.

Numbers representable by n bits are those less than 2^n. Therefore, with 512 bits, you can count up to 2^512, which is 1.340780793×10^154. So, the prime numbers which are presentable by 512 bits and by not fewer than that amount. We can approximate this by: according to Wolfram Alpha.

π (2 ^ 512) - π (2 ^ 511) ≈ 21.06094 × 10 ^ 153

Sources:

1- https://www.researchgate.net/post/What_is_the_probability_of_choosing_two_random_values_from_q_being_the_same_when_q_is_a_large_prime

2- https://primes.utm.edu/howmany.html

3- https://www.thoughtco.com/probability-of-randomly-choosing-prime-number-3126592

4- https://math.stackexchange.com/questions/263588/how-many-all-prime-numbers-p-with-length-of-bits-of-p-1024-bits

5- http://www.loyalty.org/~schoen/rsa/

So, what is the probability of the we get same prime number would be used in two different keys?

“The two primes that go into a 1024-bit RSA key are generally both 512 bits long. (If you multiply a j-digit number by a k-digit number, you can expect the answer to be around j+k digits long. Likewise with a j-bit and a k-bit number. This is based on the idea that $b^j \times b^k = b^{(j+k)}$.”

How many different 512-bit primes are there? By using the Prime Number Theorem can be used to make a good estimate. It indicates that the fraction of numbers around the size of 2^512 that are prime is around $1/(512 \ln 2)=0.0028\ldots$ or around 0.28%.

That's mean there are somewhere between 2^503 and 2^504 512-bit primes. That is a vast number, and it's incredibly implausible to envision human beings ever choosing the same one of them twice by accident.

To dive into more details we can check for this by doing the birthday paradox calculation.

Question 3. Explain why using a good RNG is crucial for the security of RSA. Provide one reference to a real world case where a poor RNG lead to a security vulnerability.

I will answer this question by describe a real life scenario of using weak random number generator, RNG and how its crucial for the security of RSA:

Suppose that you want to authenticate your self to SSH server. You have to generate brand new RSA key pair to authenticate your self when connecting to that server. This will be done through some library that implements RSA key pair let take for example (OpenSSL). The OpenSSL library will produce the key elements by generating random integers of the right length until such integers is prime number.

“The random generation uses a seed obtained from “true randomness” and then expanded into an arbitrarily long sequence of random bits with a Cryptographically secure secure pseudo random number generator.” In other word, this step is to make it not possible for an attacker to predict the bits we came up with.

Now suppose that the PRNG is poorly designed. Saying “40 bits entropy” more or less means “there are only 240 possible values for the seed or internal PRNG state”. In this case an attacker can generate all these 240 states and for each one. The attacker will try to get the private key that your application would have come up with, starting with that seed value.

After that, it will be really easy for attacker to check these keys against your public key to get your private key.

For real world case where a poor RNG lead to a security vulnerability:

This problem already happend in Debian in 2008.

Sources:

1- http://doctrina.org/How-RSA-Works-With-Examples.html

2- https://crypto.stackexchange.com/questions/35554/how-can-a-poor-rng-impact-security/35558

3- http://www.reuters.com/article/us-usa-security-nsa-rsa/exclusive-nsa-infiltrated-rsa-security-more-deeply-than-thought-study-idUSBREA2U0TY20140331