SSN LAB ASSIGNMENT: RSA

1 NOV. 2016

ALI ABDULMADZHIDOV

1.Create a 2048 bit 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 in your report.

Firstly we need to create our text file:

```
echo "Ali Abdulmadzhidov" > secret.txt
```

Then we generate 2048 bit keys private key:

```
> openssl genrsa -out private.pem 2048
Generating RSA private key, 2048 bit long modulus
.............................................................+++
..................................................................................................+++
> cat private.pem
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAyY9xqJ2oKovCqrAtniiCuI+Fb2ACoWNY/PoF/cJxr6j7gw3a
UEG9qqspZF7M3IXSdYa8L0ysT3WKqaFwLUfBfq9WAsp3YFobVqCizWWX4CZBmVp2
yhnNvPciG4BUwd/v4DNkOrXhtA0TsCqCBNfCtJE0JAkN1Y2nifRPq2Mna4WDOt3Q
QtEkp4RAko83V30/CyvSDeWT04uv7YIuvacfgSG32GZ0LzDiNE2tPPFxPDzxF7h1
vORzaN/1IBhJfqvLBWzjjaYrs18DSuH1TsZqpYf/wb+EAevL5tT6QEf0J5NSJ/iG
Hez2mW56jLVeTLTFFxIfd68Hhxx2h2H2dd/gAwIDAQABAoIBADGjjg5EpYG04cA/
mzx0JZ1mWypOrtcoLGZA/PC9zlnAeBCSfM1/I0LA3HTHlBRLMMWY5QeRDm6TOA8G
nyXHojOc4sXVLwkfp9Pxw6Zp+rlNMqKd0Nt891ouUswLEtScXesNPQXofSlmfw7H
qTWgHWdyyzCEuk95j2U+MdC3nqGv00PU8w2fKD1hGD1NUm8RQfe4uOFQqKG6J6mz
OucSlpwKjJ2lYNd7afNfTdrYaKawe68IVjQF7rpJUHIXKF446yyrebPE32fR7bGg
8qLrJ1I8lFn315AEhTeyr8csUfWYJduqsYmB7011Eizxz9JDt/4EzglUSpSVT0FT
LH6K+CECgYEA/lYLQr65ojAcutCy27T3ImCg+YxFKDaYJZ09rw1PsYqBzZV67LPg
ORz4eOjSJEc6Ai4UxWX5XJIfxOhiF08DdfvaUSUGGWen5f/F8KdysPo1oTe3tCSw
Ya1qQfuW+DyRaQPA0JSoGTe78oHtpXwJ/TaiMUCZJlj34WQ/q4fn32UCgYEAyuED
K1N1QGls476ZrnzXr4aGxpUkrAkEZQKg7apq4WQs8Uz55EmyeEIC5Z91d3q8N9GH
sKaqlepawpgneDUX6Na2EDtZJzi5X9qkrJgipZk91ebpJNGvDimOo01Re0jvlXXt
A/Nrrh/2bFkif9clUgg0WtzEOj4yOcxdxm3yD0cCgYEAoi2/1HnF6XQAfeeihZLp
m9Q+EaKuXqwhbKrNsJguXynRB4Tv84vnyMN+dasQhO2eQdGckRsrXSoYc/kyw2Yw
8MT+O4v+TajZWUI1t/Uun0eNdxYOYCKdDkwEW7rQU85bcrlf9CE2542wnTzYixOD
UyHJXtqwTdGhjunb58crxjkCgYAZxwNHEwCX4Y1TVpliQTyKcdZYDwUs0qx7RR07
dWbFIpxPQI+TwQVrIEAL+vnK545YrUfHWzXbfH5xsTF6pYxXKkjRwSivwhZTpo07
3MyyK8lej0taQUYzz3XvK9jJiFfgrq4hzWEGK8t8ssqeYbC9Pouga+KG3/Yzv9K+
bqPN6wKBgEjC36YPfuYCwJkDc+1nIEp60dcZyvBnbaGYylknxFcsE2M2UhtUCU48
TOuQ0uqeMiiP1Jmp9F4RhRIEuT/VgnnZ+Ml1Ma/9yyTpB9ew853A3wwSNgulVoyC
aCzDbpt/sUIWx8eUM+ZOvyc7SJq/E1F0364R7FmTh+XHFqX/qKNb
-----END RSA PRIVATE KEY-----
```

We generate public from private:

```
> openssl rsa -in private.pem -pubout -out public.pem
writing RSA key
> cat public.pem
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAyY9xqJ2oKovCqrAtniiC
uI+Fb2ACoWNY/PoF/cJxr6j7gw3aUEG9qqspZF7M3IXSdYa8L0ysT3WKqaFwLUfB
fq9WAsp3YFobVqCizWWX4CZBmVp2yhnNvPciG4BUwd/v4DNkOrXhtA0TsCqCBNfC
tJE0JAkN1Y2nifRPq2Mna4WDOt3QQtEkp4RAko83V30/CyvSDeWT04uv7YIuvacf
gSG32GZ0LzDiNE2tPPFxPDzxF7h1vORzaN/1IBhJfqvLBWzjjaYrs18DSuH1TsZq
pYf/wb+EAevL5tT6QEf0J5NSJ/iGHez2mW56jLVeTLTFFxIfd68Hhxx2h2H2dd/g
AwIDAQAB
-----END PUBLIC KEY-----
```

Encrpyting:

```
> cat secret.txt | openssl rsautl -encrypt -pubin -inkey public.pem > cipher.txt
> cat cipher.txt
}�•ͅ�^�y��a��N���"�I^sr��w�=(�u0?��j�HF•>��oS������%�a��:I�$����`g\�Z•,m4�|v���

  ��
•c�|���•5sT7�>C1v•j��IAj�gY��•0��••5/•����Xh�e�C�e""••�•��JX•2�Z�n�~�•��!J�����H����l%�R�h�TŤ"•,�H����2G��M•��&�{"•��h��J��P`��KI����•tXbؤ•�U��u�d�l.%
```

Decryping:

```
> cat cipher.txt | openssl rsautl -decrypt -inkey private.pem > plain.txt
> cat plain.txt
Ali Abdulmadzhidov
```

Public modulus and exponent:

```
> openssl rsa -pubin -inform PEM -text -noout < public.pem
Public-Key: (2048 bit)
Modulus:
    00:c9:8f:71:a8:9d:a8:2a:8b:c2:aa:b0:2d:9e:28:
    82:b8:8f:85:6f:60:02:a1:63:58:fc:fa:05:fd:c2:
    71:af:a8:fb:83:0d:da:50:41:bd:aa:ab:29:64:5e:
    cc:dc:85:d2:75:86:bc:2f:4c:ac:4f:75:8a:a9:a1:
    70:2d:47:c1:7e:af:56:02:ca:77:60:5a:1b:56:a0:
    a2:cd:65:97:e0:26:41:99:5a:76:ca:19:cd:bc:f7:
    22:1b:80:54:c1:df:ef:e0:33:64:3a:b5:e1:b4:0d:
    13:b0:2a:82:04:d7:c2:b4:91:34:24:09:0d:d5:8d:
    a7:89:f4:4f:ab:63:27:6b:85:83:3a:dd:d0:42:d1:
    24:a7:84:40:92:8f:37:57:7d:3f:0b:2b:d2:0d:e5:
    93:d3:8b:af:ed:82:2e:bd:a7:1f:81:21:b7:d8:66:
    74:2f:30:e2:34:4d:ad:3c:f1:71:3c:3c:f1:17:b8:
    75:bc:e4:73:68:df:f5:20:18:49:7e:ab:cb:05:6c:
    e3:8d:a6:2b:b3:5f:03:4a:e1:f5:4e:c6:6a:a5:87:
    ff:c1:bf:84:01:eb:cb:e6:d4:fa:40:47:f4:27:93:
    52:27:f8:86:1d:ec:f6:99:6e:7a:8c:b5:5e:4c:b4:
    c5:17:12:1f:77:af:07:87:1c:76:87:61:f6:75:df:
    e0:03
Exponent: 65537 (0x10001)
```

2. Assuming that you are generating a 1024 bit 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?

I used Wolfram to count prime numbers less or equeal 512 bit with prime counting function. It gave me 3.778×10^{151}. Assuming that we have already took two prime factors for first key generation, probability of pickin same factor secondly is $2/(3.778x10^{151})$

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

RNG uses some seed that is obtained from "true randomness" and then expanded into an enough long sequence of random bits. But if RNG is badly designed or entropy is too low, attacker can generate all set of "random" numbers and get all RSA keys for them. This happened in Debian on 13th May, 2008.

> On May 13th, 2008 the Debian project announced that Luciano Bello found an interesting vulnerability in the OpenSSL package they were distributing. The bug in question was caused by the removal of the following line of code from *md_rand.c*
>
> ```
>         MD_Update(&m,buf,j);
>         [ .. ]
>         MD_Update(&m,buf,j); /* purify complains */
> ```
>
> These lines were removed because they caused the Valgrind and Purify tools to produce warnings about the use of uninitialized data in any code that was linked to OpenSSL. You can see one such report to the OpenSSL team here. Removing this code has the side effect of crippling the seeding process for the OpenSSL PRNG. Instead of mixing in random data for the initial seed, the only "random" value that was used was the current process ID. On the Linux platform, the default maximum process ID is 32,768, resulting in a very small number of seed values being used for all PRNG operations.

4. Here you can find the modulus (public information) of two related 1024bit RSA keys. Your keys are numbered using the list at http://188.130.155.61/ssn/Instructions.txt. Your task is to factor them i.e. retrieve p an q. You may use any tools for this. Explain your approach. Hints: study the RSA algorithm. What private information can two keys share? What practical attacks exist? You may have to write code or use existing code for simple arithmetic operations.

After some search i thought that those two modules can have one of the prime factors same. If we know that, we can compute gcd of them and find the second factor by dividing that modulus to that gcd.

I used python gcd function from factorize package.

```
> python

> from factorize import gcd

> n1 =
0xe9942c4b4d6ac25c231feeb24b6f4b1693eaae4f97ba88e5c694f5fbfc407e92d9472103
5701cac023779a7f8194f09d0cb789ecfc5295234ef8aa62af446205223aa76760fe307ee1
ab4e673e4c8b6bfa73a7e147e1fc606808efc5d666167a243c15cd82649d2bfe9ebcec01a9
7174b795d8cc6fb16c85a780ad70b23081d5L

> n2 =
0xcefd3933ca8301944de8a257775e548aa24ed0841bbbbe856b0f135478915b84269b0327
620e0f4e7c9ccad67cd8120cd33e7d6a7e95a5048a49a113a0139c9411a77aa88bf1adea12
535bf62b4d8c6187ea51e34f79c746e12e014cd20d6f6856086682763ef99ace1415b795b9
a6b433a2a77d97cf6e574e68bb79e6d2d935L

> p = gcd(n1,n2)
> q1 = n1/p
> q2 = n2/p
> p
12336923788090286356293627535978275509551220466238172479246793216159360335
42884789662468780030454854196935209049766068525844065628387817177582276166
3689487L
> q1
13295422330708449110067252512639224991344905545469973118546306846289075340
31899870744033638973790941339408564316138718306198475979600667871433831038
2464219L
> q2
11781926318507083481218847334007109530409514414858167505452121203204186409
12596092853518674026953916311391134316954061481996926952393706320179808417
6338811L
```

5. Now that you have the p and q for both keys, recreate the first public and private key using this script. Encrypt your name with the private key and post the public key and the base64 encrypted data in your report.

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDplCxLTWrCXCMf7rJLb0sWk+qu
T5e6iOXGlPX7/EB+ktlHIQNXAcrAI3eaf4GU8J0Mt4ns/FKVI074qmKvRGIFIjqn
Z2D+MH7hq05nPkyLa/pzp+FH4fxgaAjvxdZmFnokPBXNgmSdK/6evOwBqXF0t5XY
zG+xbIWngK1wsjCB1QIDAQAB
-----END PUBLIC KEY-----
M4Fe3urZRufArJOA1c0/NIyV0SiGh2Dnse7uysLM+WjNTTbYPLsHYM1jWbaIgy8EP5E2wB7485IvbHqu
tjlySFlG5feRgue+U7rA3ABLkq4uzK4bfokxfThbaUX37KQ3WbjIoNNDs1z3DCetVqLDoocpp2y4c/Q4
0NuwHHDmZts=
```