

از طریق سایت زیر یک کلید 256 بیتی تولید کردیم:

The screenshot shows the 'RSA Key Generator' interface on the website [csfieldguide.org/nz/en/interactives/rsa-key-generator/](https://csfieldguide.org/nz/en/interactives/rsa-key-generator/). The 'Key Size' is set to '256 bits' and the 'Format Scheme' is 'PKCS #8 (base64)'. The 'Generate' button has been clicked, resulting in the following keys:

**Public Key:**

```
-----BEGIN PUBLIC KEY-----
MDwvDQYJKoZIhvcNAQEBBQADKwAwKAIhA0FBpjo1encU0ev2j9fFHASIBIUAPO2f
7Q0ba+n4BL2TAqMBAAE=
-----END PUBLIC KEY-----
```

**Private Key:**

```
-----BEGIN PRIVATE KEY-----
MIHCAGeAMAGCSqGSIb3DQERBAUABIG1MIgAgEAAiEA4UGe0jV6dx056/aP18Uc
BIqEh0A/TZ/tA5tR6fjwvZHCawEAAQIgrFRmEqIo6aXC2T3rVQ1cv1wzIVZUSON
EBfdP87IsseCE007m7Cx/tqL/C3706MDvjDHAhEASTA04hxH033RvSCOF25Y1QIR
AMG8hRbZLTca1GJCymVRfscEG0j9LsMbLuFQ1Iw14nWTPUCECvETTfGLpVVOLq
7P3Y+yc=
-----END PRIVATE KEY-----
```

از طریق انجام مراحل زیر یک پیام را توسط کلید عمومی رمز کردیم.

```
mohammad@mohammad-X556UQ ~ -/Documents/anniat/az/hw2/q1 $ echo hello from Iran > msg.txt
mohammad@mohammad-X556UQ ~ -/Documents/anniat/az/hw2/q1 $ ls
keys.txt  msg.txt
mohammad@mohammad-X556UQ ~ -/Documents/anniat/az/hw2/q1 $ cat msg
cat: msg: No such file or directory
mohammad@mohammad-X556UQ ~ -/Documents/anniat/az/hw2/q1 $ cat msg.txt
hello from Iran
mohammad@mohammad-X556UQ ~ -/Documents/anniat/az/hw2/q1 $ openssl rsautl -encrypt -pubin -inkey
rsautl: Option -inkey needs a value
rsautl: Use -help for summary.
mohammad@mohammad-X556UQ ~ -/Documents/anniat/az/hw2/q1 $ gedit public.pem
mohammad@mohammad-X556UQ ~ -/Documents/anniat/az/hw2/q1 $ openssl rsautl -encrypt -pubin -inkey public.pem -in msg.txt -out enc-msg.txt
mohammad@mohammad-X556UQ ~ -/Documents/anniat/az/hw2/q1 $ ls
enc-msg.txt  keys.txt  msg.txt  public.pem
mohammad@mohammad-X556UQ ~ -/Documents/anniat/az/hw2/q1 $ cat enc-msg.txt
mohammad@mohammad-X556UQ ~ -/Documents/anniat/az/hw2/q1 $
```

در مرحله بعد لازم بود تا n را محاسبه کنیم.

$n = 101886383249647936993768038558632029819971268904921994147982510997493202861459$

برای بدست آوردن n مراحل زیر را طی کردیم:

```

mohammad@mohammad-X556UQ ~/Documents/amniat/az/hw2/q1 openssl rsa -pubin -inform PEM -text -noout < public.pem
RSA Public-Key: (256 bit)
Modulus:
  00:e1:41:a6:3a:35:7a:77:14:39:eb:f6:8f:d7:c5:
  1c:04:88:04:85:00:3f:4d:9f:ed:03:9b:6b:e9:f8:
  f0:bd:93
Exponent: 65537 (0x10001)
mohammad@mohammad-X556UQ ~/Documents/amniat/az/hw2/q1 python
Python 2.7.18rc1 (default, Apr 7 2020, 12:05:55)
[GCC 9.3.0] on linux2
Type "help", "copyright", "credits" or "license()" for more information.
>>> s = '00:e1:41:a6:3a:35:7a:77:14:39:eb:f6:8f:d7:c5:1c:04:88:04:85:00:3f:4d:9f:ed:03:9b:6b:e9:f8:f0:bd:93'
>>> s.replace(':')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: replace() takes at least 2 arguments (1 given)
>>> s.replace(':', '')
'00e141a63a357a771439ebf68fd7c51c04880485003f4d9fed039b6be9f8f0bd93'
>>> s = s.replace(':', '')
>>> s.upper()
'00E141A63A357A771439EBF68FD7C51C04880485003F4D9FED039B6BE9F8F0BD93'
>>>
KeyboardInterrupt
>>>
KeyboardInterrupt
>>>
[1] + 57991 suspended python
mohammad@mohammad-X556UQ ~/Documents/amniat/az/hw2/q1 bc -q
ibase=16
00E141A63A357A771439EBF68FD7C51C04880485003F4D9FED039B6BE9F8F0BD93
10188638324964793699376803855863202981997126890492199414798251099749\
3202861459

```

سپس از طریق yafu عدد به دست آمده را فکتور کردیم و نتایج حاصل در فایل factor.log ذخیره شده. سپس از طریق دو فاکتور به دست آمده به private key رسیدیم.

```

Command Prompt
fac: no tune info: using qs/gnfs crossover of 95 digits
div: primes less than 10000
fat: 1000000 iterations
rho: x^2 + 3, starting 1000 iterations on C78
rho: x^2 + 2, starting 1000 iterations on C78
rho: x^2 + 1, starting 1000 iterations on C78
pm1: starting B1 = 150K, B2 = gmp-ecm default on C78
ecm: 30/30 curves on C78, B1=2K, B2=gmp-ecm default
ecm: 74/74 curves on C78, B1=11K, B2=gmp-ecm default
ecm: 161/161 curves on C78, B1=50K, B2=gmp-ecm default,
starting SIQS on c78: 1018863832496479369937680385586320
---- sieving in progress (1 thread): 36224 relations r
---- Press ctrl-c to abort and save state
36313 rels found: 18214 full + 18099 from 191886 partial
SIQS elapsed time = 107.3199 seconds.
Total factoring time = 129.5904 seconds
***factors found***
p39 = 304643513883571219680513378958891587797
p39 = 334444616761435189484191904554530255047
ans = 1
C:\Users\HO3EIN\Desktop>

```

cryptool.org/en/cto-highlights/rsa-step-by-step

$n = p \times q$  101886383249647936993768038558632029819971268904921994147982510997493202861459 (256 Bit)

## Public key

The product  $n$  is also called module in the RSA method. The public key consists of the module  $n$  and an exponent  $e$ .

$e$

This  $e$  may even be pre-selected and the same for all participants.

## Secret key

RSA uses the Euler  $\phi$  function of  $n$  to calculate the secret key. This is defined as

$\phi(n) = (p - 1) \times (q - 1)$  101886383249647936993768038558632029819332180774276967738817805713979781018616

Here it is used that  $p$  and  $q$  are different. Otherwise, the  $\phi$  function would calculate differently.

It is important for RSA that the value of the  $\phi$  function is coprime to  $e$  (the largest common divisor must be 1).

$\gcd(e, \phi(n))$  1

To determine the value of  $\phi(n)$ , it is not enough to know  $n$ . Only with the knowledge of  $p$  and  $q$  we can efficiently determine  $\phi(n)$ .

The secret key also consists of  $n$  and a  $d$  with the property that  $e \times d$  is a multiple of  $\phi(n)$  plus one.

Expressed in formulas, the following must apply:

$e \times d \equiv 1 \pmod{\phi(n)}$

In this case, the mod expression means equality with regard to a residual class. It is  $x \equiv y \pmod{z}$  if and only if there is an integer  $a$  with  $x - y = z \times a$ .

For the chosen values of  $p$ ,  $q$ , and  $e$ , we get  $d$  as:

$d$  30906225475731281374431368639785232049198525318409852697677616881973817029313

This website uses cookies to ensure you get the best experience on our website. [Cookie policy](#) [Privacy policy](#)

و در مرحله آخر کلید را به فرمت مد نظر تبدیل کرده و در فایل private-key.pem ذخیره کرده و با استفاده از کدهای زیر پیام را رمزگشایی کردیم.

```
mohammad@mohammad-X556UQ: ~/Documents/amniat/az/hw2/q1
mohammad@mohammad-X556UQ ~$ openssl rsautl -decrypt -inkey private-key.pem -in enc-msg.txt -out dec-msg.txt
mohammad@mohammad-X556UQ ~$ cat dec-msg.txt
hello from Iran
mohammad@mohammad-X556UQ ~$
```

قطعا با افزایش طول کلید زمان لازم برای بایپس کردن آن و فکتور کردن چندین برابر خواهد شد.

## سوال دوم

برای این منظور از کتابخانه pycryptodome استفاده کردیم. ابتدا کتابخانه را نصب کردیم. سپس در مرحله اول یک کلید عمومی از طریق این کتابخانه برای سرور ایجاد کرده و سپس کلاینت که ارتباط برقرار میکند کلید برای او

ارسال میشود.

```
#generate public key for server
random_generator = Random.new().read
RsaKey = RSA.generate(1024, random_generator)
print('Key generated')
serverPublicKey = RsaKey.publickey().n

#send public key to client
sending_data = input("Press Enter To Send PublicKey For Client")
conn.send(bytes(str(serverPublicKey).encode()))
```

سپس کلاینت این کلید را دریافت کرده و یک کلید خصوصی تولید کرده و با این کلید رمز کرده و برای ما ارسال میکند.

```
# receive server's public key
data = ''
while 1:
    try:
        data = s.recv(1024)
        data = int(data.decode())
        break
    except ConnectionResetError:
        print('Broken PIPE!')

#send aes key for server
serverPublicKey = RSA.construct((data, 65537))
encryptor = PKCS1_OAEP.new(serverPublicKey)
sending_data = str(input("Press Enter To Send SessionKey For Client"))
sessionKey = os.urandom(64)
sending_data = encryptor.encrypt(sessionKey)
s.send(sending_data)
```

در مرحله بعد در سرور این کلید دریافت شده و رمز گشایی میشود.

```
#receive client session key
data = ''
while 1:
    try:
        data = (conn.recv(1024))
        break
    except ConnectionResetError:
        print('Broken PIPE!')

encryptor = PKCS1_OAEP.new(RsaKey)
key = encryptor.decrypt(data)
```

و ازین به بعد پیامها بین کلاینت و سرور با این کلید رد و بدل میشود.

## سوال سوم

ابتدا لیستی از دانشجوها با نام های 1 تا 149 و رمز هایی مشخص درست کردیم.

در مرحله بعد با دستور

*openssl genrsa -out key.pem 2048*

یک کلید به طول 2048 تولید کردیم سپس به رمز کردن نام های موجود در فایل data.json پرداختیم و در فایل encrypted.json ذخیره کردیم. سپس کد مربوط به سرچ را نوشتیم که ابتدا به دنبال نام دانشجو گشته و سپس در صورت یافتن آن شماره دانشجویی مربوطه را رمزگشایی می کند. خطر احتمالی میتواند در صورتی باشد که نام دو دانشجو یکی باشد در این صورت در خروجی کد دو شماره دانشجویی خواهیم داشت.