Generating random prime with n bits

[Back] Many operations in public key encryption involve the $\pmod{p}$ operation and where we take the modulus of a prime number. This is defined as a finite field. In this case we will use Python to generate a random n-bit prime number. In RSA, for example, we take two prime numbers (p and q) and then multiply them together to create a modulus (N). The value of N is then part of the public and the private key. For RSA-2048 we use two 1,024-bit prime numbers, and RSA-4096 uses two 2,048-bit prime numbers. In ECC (Elliptic Curve Cryptography) we typically use much smaller prime numbers, such as with 160-bit prime numbers (and which gives the equivalent security to RSA-1024) and 256-bit prime numbers (for Bitcoin addresses and signatures). Diffie-Hellman Group 1 uses a 768-bit modulus, Group 2 uses a 1024-bit modulus, Group 5 uses a 1,536-bit modulus, and Group 14 uses a 2,048-bit modulus.

Number of bits in prime number (n):
16-bit

Determine

Note: RSA-2048 will take a few seconds to compute.

```
No of bits (n) in prime is  16

Random n-bit Prime (p):  63059
Random n-bit Prime (q):  59377

N=p*q= 3744254243

PHI (p-1)(q-1)= 3744131808

e= 65537
d= 3225848033

Count of decimal digits (p):  5
Count of decimal digits (N):  10


=== Let's try these keys ==
RSA Message:  5
RSA Cipher(c=M^e mod N):  1969852118
RSA Decipher (c^d mod N):  5
```

Outline

Many operations in public key encryption involve the $\pmod{p}$ operation and where we take the modulus of a prime number. This is defined as a finite field. In this case we will use Python to generate a random n-bit prime number. In RSA, for example, we take two prime numbers (p and q) and then multiply them together to create a modulus (N). The value of N is then part of the public and the private key. For RSA-2048 we use two 1,024-bit prime numbers, and RSA-4096 uses two 2,048-bit prime numbers.

In the following Python program we will generate two random prime numbers (p and q) and which are a given length long. We will then use the RSA method of finding the modulus (N) and determine the number of decimal digits in these values:

```
import Crypto.Util.number

import sys

bits=100

if (len(sys.argv)>1):
        bits=int(sys.argv[1])

print ("No of bits in prime is ",bits)

p=Crypto.Util.number.getPrime(bits, randfunc=Crypto.Random.get_random_bytes)
print ("\nRandom n-bit Prime (p): ",p)

q=Crypto.Util.number.getPrime(bits, randfunc=Crypto.Random.get_random_bytes)
print ("\nRandom n-bit Prime (q): ",q)

N=p*q

print ("\nN=p*q=",N)

PHI=(p-1)*(q-1)
```

```
print ("\nPHI (p-1)(q-1)=",PHI)

e=65537
print ("\ne=",e)
d=Crypto.Util.number.inverse(e,PHI)
print ("d=",d)

print ("\nCount of decimal digits (p): ",len(str(p)))
print ("Count of decimal digits (N): ",len(str(N)))

M=5
print ("\n\n=== Let's try these keys ==")
print ("\nRSA Message: ",M)
enc=pow(M,e,N)
print ("RSA Cipher(c=M^e mod N): ",enc)
dec = pow(enc,d,N)
print ("RSA Decipher (c^d mod N): ",dec)
```

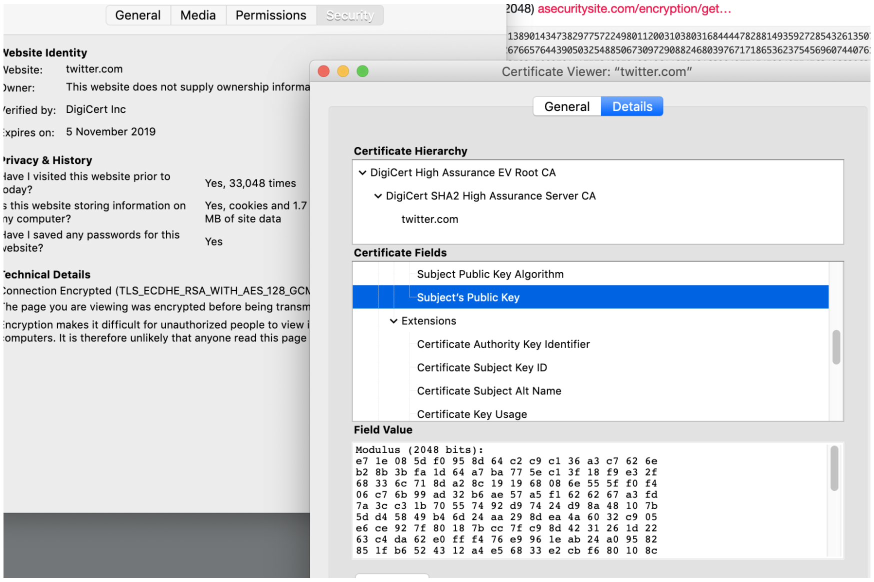
The following is a sample run and have p and q have 256 bits and N has 512 bits. We can see the number of decimal digits in N is 154:

```
No of bits in prime is  256

Random n-bit Prime (p):  66879465661348111229871989287968040993513351195484998191057052014006844134449

Random n-bit Prime (q):  109939025753834733498749075564102728424911782303658486825359178646821371085889

N=p*q= 735266329774565570756477089897302611112357113167434206427431046665668214987571371319061959385762263522955035069584042370775

PHI (p-1)(q-1)= 735266329774565570756477089897302611112357113167434206427431046665668214987553689469920441101289401416469827992642

e= 65537
d= 6819755922304426102747146323998250426734036016007571769550673822980297203785407650915979348036083257381776925124546653452752122

Count of decimal digits (p):  77
Count of decimal digits (N):  154


=== Let's try these keys ==

RSA Message:  5
RSA Cipher(c=M^e mod N):  37373004143418966879827042746409763351778674146107659372531113934444488211020934370878779056375225862122
RSA Decipher (c^d mod N):  5
```

Unfortunately for RSA-2048-which uses two 1,024 bit prime numbers- would take too long to generate with the on-line tool, so here's a sample run:

```
No of bits in prime is  1024

Random N-bit Prime (p):  149266604066765214257465899845052595936980433085281120472438633560109109845062080813195674897136525949840

Random N-bit Prime (q):  116136133237524628623079973436761666157812135802554422133884399716278215827708188540430994158743163224360

N=p*q= 173352462178106804995652823641302823479136944111397065523376469699967951853105399726952135895219488778887101481083141833224

PHI (p-1)(q-1)= 173352462178106804995652823641302823479136944111397065523376469699967951853105399726952135895219488778887101481083

e= 65537
d= 1568893081643964314006920659877106728873468356207835901458417160888383528123539462289197252903421508146868280535801820582885959

Count of decimal digits (p):  309
Count of decimal digits (N):  617


=== Let's try these keys ==

RSA Message:  5
RSA Cipher(c=M^e mod N):  50066111225230104272225430069095729510385148035918957968323694564616506415687318093391218302118942032523
RSA Decipher (c^d mod N):  5
```

We can see, in this case, that we have 617 decimal digits for the RSA modulus, and which are generated from two 1,024 bit prime numbers. Overall RSA-2048 is used fairly extensively within digital certificates and in creating TLS tunnels. Here is an example of a connection to Twitter, and where we see that the Modulus is 2,048 bits long (RSA-2048):



Run ▶ open in replit

main.py ×

```
# https://repl.it/@billbuchanan/getprimen
import Crypto.Util.number

import sys

bits=100

if (len(sys.argv)>1):
```

Console Shell

```
Python 3.8.2 (default, Feb 26 2020, 02:56:10
```

Presentation

Follow @billatnapier Tweet Tweet #Asecuritysite

Ref: William J Buchanan (2021), *Generating random prime with n bits*, Asecuritysite, from https://asecuritysite.com/encryption/getprimen