Programming Assignment 4[1]

Your goal in this project is to break RSA when the public modulus N is generated incorrectly. This should serve as yet another reminder not to implement crypto primitives yourself.

Normally, the primes that comprise an RSA modulus are generated independently of one another. But suppose a developer decides to generate the first prime p by choosing a random number R and scanning for a prime close by. The second prime q is generated by scanning for some other random prime also close to R. We show that the resulting RSA modulus $N = p \cdot q$ can be easily factored.

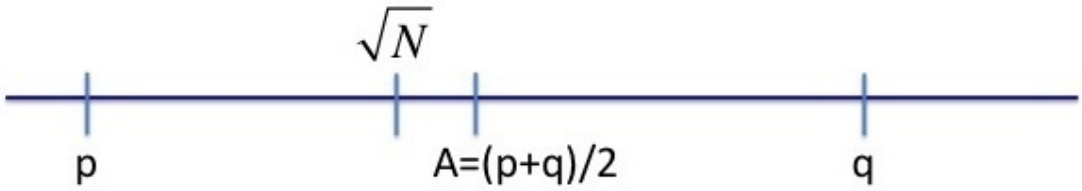
Suppose you are given a composite N and are told that N is a product of two relatively close primes p and q, namely p and q satisfy

$$|p - q| < 2N^{1/4} \tag{1}$$

Your goal is to factor N.

Let A be the arithmetic average of the two primes, that is $A = \frac{p+q}{2}$. Since p and q are odd, we know that $p + q$ is even and therefore A is an integer.

To factor N you first observe that under condition (1) the quantity $\sqrt{N}$ is very close to A. In particular $A - \sqrt{N} < 1$ as shown below. But since A is an integer, rounding $\sqrt{N}$ up to the closest integer reveals the value of A. In code, A = ceil(sqrt(N)) where "ceil" is the ceiling function. Visually, the numbers $p, q, \sqrt{N}$ and A are ordered as follows:



Since A is the exact mid-point between p and q there is an integer x such that $p = A - x$ and $q = A + x$. But then

$$N = pq = (A - x)(A + x) = A^2 - x^2$$

and therefore $x = \sqrt{A^2 - N}$ Now, given x and A you can find the factors p and q of N since $p = A - x$ and $q = A + x$.

In the following challenges, you will factor the given moduli using the method outlined above. To solve this assignment it is best to use an environment that supports multi-precision arithmetic and square roots. In Python you could use the gmpy2 module. In C you can use GMP.

1 Question 1

Factoring challenge #1: The following modulus N is a products of two primes p and q where $|p - q| < 2N^{1/4}$. Find the smaller of the two factors and enter it as a decimal integer.

[1]Source: https://class.coursera.org/crypto-014/

```
N = 17976931348623159077293051907890247336179769789423065727343008115 \
    77326758055056206869853794492129829595855013875371640157101398586 \
    47833778606925583497541085196591615128057575940752635007475935288 \
    71082364994994077189561705436114947486504671101510156394068052754 \
    00715845608785776637430400863407428552785490925 81
```

2 Question 2

Factoring challenge #2: The following modulus N is a products of two primes p and q where $|p-q| < 2^{11}N^{1/4}$. Find the smaller of the two factors and enter it as a decimal integer. *Hint:* in this case $A - \sqrt{N} < 2^{20}$ so try scanning for A from $\sqrt{N}$ upwards, until you succeed in factoring N.

```
N = 648455842808071669662824265346772278726343720706976263060439070378 7 \
    97308618081116462714015276061417569195587321840254520655424906719 89 \
    24288448418393532819729885313105117386489659625828215025049902644 52 \
    10088528167330371114229642102784028930765745864523368335707783468 97 \
    15838646088239640236866252211790085787877
```

3 Question 3

Factoring challenge #3: The following modulus N is a products of two primes p and q where $|3p-2q| < N^{1/4}$. Find the smaller of the two factors and enter it as a decimal integer. *Hint:* use the calculation below to show that $\sqrt{6N}$ is close to $\frac{3p+2q}{2}$ and then adapt the method above to factor N.

```
N = 720062263747350425279564435525583738338084451473999841826653057981 91 \
    63556901883377904234086641876639384851752649940178970835240791356868 \
    77441155132015188279331812309091996246361896836573643119174094961348 \
    52463970788523879939683923036467667022162701835329944324119217381272 \
    92761475307485973021927513757393879 29
```

The only remaining mystery is why $A - \sqrt{N} < 1$. This follows from the following simple calculation. First observe that

$$A^2 - N = \left(\frac{p+q}{2}\right)^2 - N = \frac{p^2+2N+q^2}{4} - N = \frac{p^2-2N+q^2}{4} = (p-q)^2/4$$

Now, since for all $x, y:\ \ (x-y)(x+y) = x^2 - y^2$ we obtain

$$A - \sqrt{N} = (A-\sqrt{N})\frac{A+\sqrt{N}}{A+\sqrt{N}} = \frac{A^2-N}{A+\sqrt{N}} = \frac{(p-q)^2/4}{A+\sqrt{N}}$$

and since $\sqrt{N} \le A$ it follows that

$$A - \sqrt{N} \le \frac{(p-q)^2/4}{2\sqrt{N}} = \frac{(p-q)^2}{8\sqrt{N}}$$

By assumption (1) we know that $(p-q)^2 < 4\sqrt{N}$ and therefore

$$A - \sqrt{N} \leq \frac{4\sqrt{N}}{8\sqrt{N}} = 1/2$$

as required.

Further reading: the method described above is a greatly simplified version of a much more general result on factoring when the high order bits of the prime factor are known.

4 Question 4

The challenge ciphertext provided below is the result of encrypting a short secret ASCII plaintext using the RSA modulus given in the first factorization challenge. The encryption exponent used is $e = 65537$. The ASCII plaintext was encoded using PKCS v1.5 before the RSA function was applied, as described in Lecture 11.4.

Use the factorization you obtained for this RSA modulus to decrypt this challenge ciphertext and enter the resulting English plaintext in the box below. Recall that the factorization of N enables you to compute $\varphi(N)$ from which you can obtain the RSA decryption exponent.

Challenge ciphertext (as a decimal integer):

```
220964518674103817763065611348834180174100697878928310717318391436761356001 20 \
538004282329650473509424343946219751512256465839967942889460764542040581564 74 \
898801373486412045232522932017648791666640299750918872997169052608322206777 16 \
000193292608700095799937240774589677736978175712672299511486629596279347915 40
```

After you use the decryption exponent to decrypt the challenge ciphertext you will obtain a PKCS1 encoded plaintext. To undo the encoding it is best to write the decrypted value in hex. You will observe that the number starts with a '0x02' followed by many random non-zero digits. Look for the '0x00' separator and the digits following this separator are the ASCII letters of the plaintext. (note: the separator used here is '0x00', not '0xFF' as stated in the lecture)