Ques 3 - Cracking RSA

Algorithm:

Factorization:

1. For this, I first used Lenstra's algorithm for elliptic curve based integer factorization. However, with that I was able to factorize only upto 121421362624460336659080001 (number 93). The code for the same is present in the lenstra.py. Depending upon the input, the value of “limit” will need to be set. A large value of “limit” was not feasible on my machine because of memory issues (since I have used sieve for checking the primes upto limit in time O(limit)).
2. But, since later in the class it was mentioned that we can use libraries for factorization as well, I have used the sympy library. After this, I’m able to factorize all the integers given in nlist.txt.

Keys:

1. I now have the factorization of n. If the number of factors is not equal to 2, I stop. If n is a square (and so the 2 factors are the same), I stop. Else, I get two primes p and q such that n = p*q.
2. phi = (p-1)*(q-1)
3. For public key, I choose a random integer e in [0, phi-1] such that gcd(e, phi) = 1.
4. For private key, I calculate the multiplicative inverse of e, that is, d = e^(-1) mod phi, where d is the private key and e is the public key.

Encryption & Decryption:

1. Suppose I have an integer M (we are given a text, and the steps to convert this text to integer are described later). Its ciphertext C will be M^e mod n. For the ciphertext C, the plaintext M will be C^d mod n.
2. Since e and d can be huge, for calculating these large powers, I have used the fast exponentiation method.

Text to Number:

1. In encryption, we will need an integer, so we need a way to convert text to number.
2. Say n = 3899666309221, and text = “haha, I..”.
3. The ascii value of “h” is 104 and of “a” is 97. I have concatenated the ASCII values until the number remains lesser than n. This is because in encryption and decryption, we take mod n, and if we take message M > n, we won’t be able to find it back. For M < n, however, there can only be one choice.
4. Further, if we concatenate the values straightaway, we may lose some information -- how many characters were present in the string initially? For instance, if I have an integer 111, I don’t know if it’s concatenation 3 one’s or is it just one whole integer 111. In such cases, obtaining plaintext from the the integer will be an issue. To counter this,

since ASCII values lie in 0 to 255, I add additional 0s in the front of the number until the number of digits in less than 3. So, using this, the ASCII value we use for “a” will be 097.

5. As an example iteration, we have (for text and n as mentioned above):
    a. 104 < n -- continue
    b. 104097 < n -- continue
    c. 104097104 < n -- continue
    d. 104097104097 < n -- continue
    e. 104097104097044 > n -- use previous value and go to step 1.

Number to Text:

1. Since the output of RSA decryption will be an integer, we need to find a way to convert it to the appropriate text.
2. If some text starts with “a” (with ASCII value 97), its integer will start with 97. Note that in text to integer, we added an extra 0, but it will now be discarded from front because we have an integer, and not string. Thus, add one or two zero (depending upon the number of digits -- they should finally be a divisor of 3) in the front.
3. After this, break the number in chunks of length 3, obtain the characters corresponding to the ASCII values, and we have our plain text!

Why this strategy?

I have used this concatenating and splitting based strategy because:

1. As mentioned, if the integer becomes larger than n, its impossible to get back the plaintext.
2. I have **NOT** worked with single integers, but their concatenation because if I process all characters separately, then it simply becomes a substitution cipher. After grouping, however, if the number n is say 30 digits long, I can work with 10 characters at once (with 52^10 possible cases, if you consider just small and capital letters, which is huge).

Results:

For the purpose of reproducibility, I have seeded the random generator with 0.

Invalid inputs:

The following values of n are invalid because they are squares (and in RSA we take p not equal to q):

1. 2605796209: 51047^2
2. 3678058609: 60647^2
3. 3694086841: 60779^2
4. 4303234442515849: 65599043^2
5. 961801438520428081: 980714759^2
6. 116517380705369133516848 89: 3413464233083^2
7. 840859136266769099141989302482624329: 916983716467620323^2
8. 131687290707360806646882657121529 4289: 114755082984310898 3^2

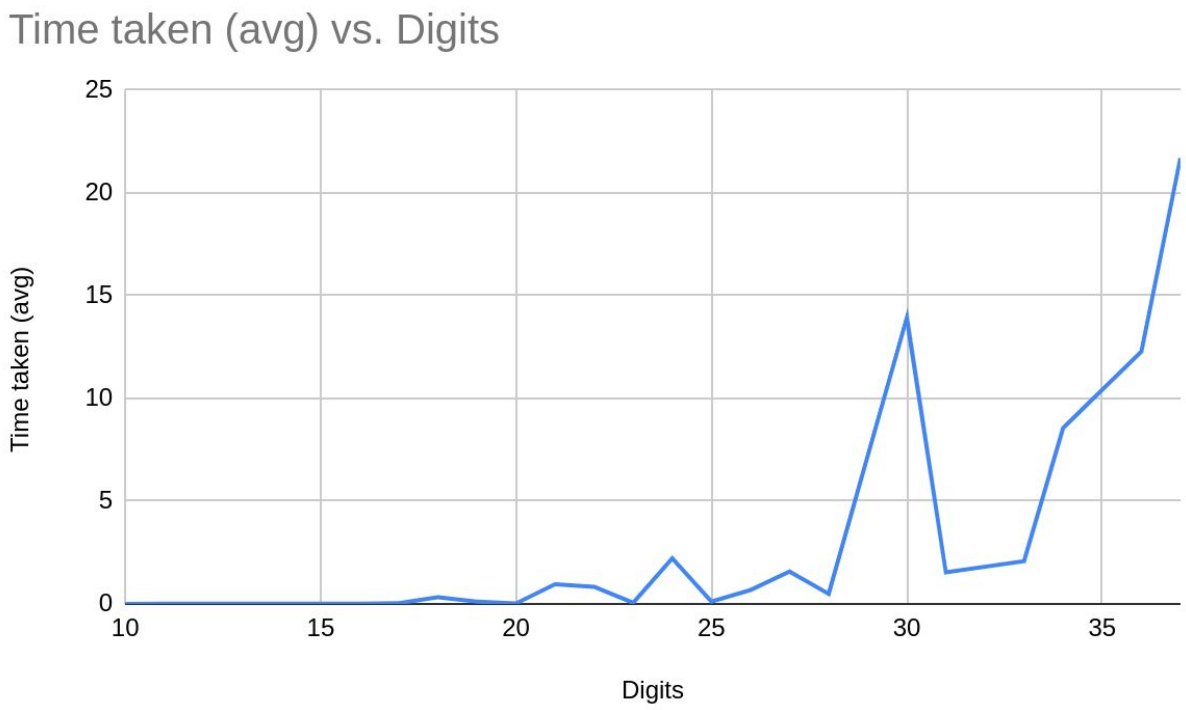
Further, the integer at position 95, 4243347351991357514018277, is invalid because it has 6 factors: 3, 89, 409, 3923, 73091, 135516383263

Plot:

For this, I take the average of the time taken for numbers with digits d, and plot them. In the submission folder, there are 2 additional files present (than were asked):

1. Time_all.pdf: Reports time for all integers present in nlist.txt
2. Time_avg.pdf: Reports average time taken for digit d for integers present in nlist.txt

The following plot is based on file #2.



The time taken, as expected, increases with increase in number of digits d. A few abnormalities are however obtained in this behaviour, and are mainly because of the fact that we don't have too many numbers for each digit, and for some of the numbers, the time taken is less while for some its huge (because time taken to factorise depends on the factors).