Assignment #9:

Implement RSA : (consists of 3 programs) 15 marks towards the grade

Strictly follow instructions.

When I say take input from command line, I mean pass the argument with the executable (and NOT interactive)

Thus for python, for program 1 the command should be (if the seed is 1728292)

$python Generate-keys.py 1728292

Talk to your friends if you have confusion.

You must write your name and roll number in the program files as comment

For c++, please write as comment the command to be passed to gnu compiler.

I would prefer if you upload all the files with names as specified to moodle. You can check that it has been submitted, time etc. If you have difficulty, then you zip them with your name as zipfile name, and send by email to

`rlkcmi+cryp@gmail.com` with Subject : CRYP: Assignment 9.

First program:

Generate-keys.cpp or Generate-keys.py

It takes a 32-bit integer as a seed from command line for the Mersenne Twister random number generator.

Outputs p, q, n, e, d (in decimal notation) on terminal (one integer per line) where

p, q are prime numbers $2^{63} < p, q < 2^{64}$ and $n = pq$, $\phi = (p-1)(q-1)$ with $2^{127} < n < 2^{128}$ and

e, d are integers such that $ed = 1 \bmod \phi$, and $2^{126} < e, d < \phi$.

Further,

The program writes n, e to a file named “Public.key.txt”, one integer per line and

The program writes n, d to a file named “Private.key.txt”, one integer per line

Some clarifications: The integers should be written as decimal numbers : For example if
$n = \mathbf{310282366920938463463374607431768211451}$ and
$e = \mathbf{296074693420933176346138211450282368467}$,
the "Public.key.txt" file should have two lines (the second line should end with end of line character)

```
310282366920938463463374607431768211451
296074693420933176346138211450282368467
```

Here these values of n and e do not satisfy the required properties. These are just to specify the writing format.

You should use Mersenne-Twister algorithm for random number generation and use the seed to initialise the same.

To generate p, q you may use :
Algorithm: Random search for a prime using the Miller-Rabin test, (HAC, p146) to generate probable primes, p, q- take $t = 25$, and $B = 1024$

or you may use Algorithm: Maurer's algorithm for generating provable primes, (HAC p153).

(I understand that since this involves generation of random numbers, even with same seed, different implementations may lead to different choices of p, q. So as long as p, q, n, e, d satisfy the required conditions it is fine.)

2: Encryption : encrypt.cpp or encrypt.py
Input "plaintext": (read from a file "plaintext.txt")
The plaintext would be a number m (assume to be less than n)
Read (n, e) form the file "Public.key.txt" and compute

$$c = m^e \bmod n.$$

Write "ciphertext" c to a file "ciphertext.txt" and also display c on screen (stdout)

Some clarifications: The encryption and decryption programmes should not require any arguments to be passed on command line or otherwise.

For encryption, the input file names: "plaintext.txt" for input and "Public.key.txt" for the key have to be hardcoded in the . The file "plaintext.txt" will have exactly one line, containing a number m. For example

290282366920938463463374607431768211451

(you may use this value or any other value, less than n as per your choice.
Likewise, the output will also be a single line, containing the number c.

The same comments apply to the decryption program.

3: Decryption : decrypt.cpp or decrypt.py Input "ciphertext": (read from a file "ciphertext.txt")
The ciphertext would be a number c (assume to be less than n)
Read (n, d) form the file "Private.key.txt"
and compute

$$k = c^d \bmod n.$$

write "decrypted-plaintext" k to a file "decrypted-plaintext.txt" and also display k on screen (stdout)

Include "plaintext.txt", "ciphertext.txt", "decrypted-plaintext" , "Public.key.txt", "Private.key.txt" in your submission. (Upload total of 8 files, 5 text files and 3 files containing cpp or py code).

The encryption and decryption programs should be inter-portable- as long as correct key files are supplied, it should work. Thus keys generated by the code written by X should work with the eccryption code written by Y and and inturn, if the ciphertext and private key as passed to code written by Z, the decrypted text should match the plaintext. This assumes that the keys are exactly as per specification.