Programming Assignment #2

RSA

---

1. Introduction

Cryptography is a key element in preserving confidentiality; it is used to scramble data such that it cannot be easily read by anyone except the intended receiver.

In this programming assignment, you will first learn how to generate large prime numbers followed by implementing the RSA algorithm.

2. Details

Since you will have to compute large exponents as part of the RSA algorithm, the standard integer types (int - 16 or 32 bits, long int - 32 bits and and even long long int - 64 bits) of most computer languages will not suffice.

Here are your choices:

- **Java** has a built-in bigint class that behaves exactly like an int, except that it is not bound by a finite value
- **C++** you'll have to use a library such as NTL (Library for doing Number Theory) or GMP (the GNU Multiple Precision Arithmetic Library)
- In **Ruby** you don't have to worry about the size of numbers at all, Ruby automatically takes care of any conversion between numbers of fixed size (Fixnum - 16, 32, 64 bits) and Bignum (numbers with an arbitrary number of digits)

3. What You Have to Do

The assignment is broken down into two main parts. Each part will be graded separately, but each part will also be reused for other parts of the assignment. This will allow you to test each part conclusively before moving on.

3.1 Prime Numbers (40%)

RSA requires finding large prime numbers very quickly.

You will need to research and implement a method for primality testing of large numbers. There are a number of such methods such as Fermat's, Miller-Rabin, AKS, etc.

Your task is to develop two programs (make sure you use your two programs to check each other), both running from the command line.

You'll get 50% of the credit for this section of the assignment if the programs work and 100% of the credit if they work *fast*, i.e. they complete in seconds for 1024-bit numbers.

The first program is called **primecheck** and will take a single argument, an arbitrarily long positive integer and return either True or False depending on whether the number provided as an argument is a prime number or not. You may not use the library functions that come with the language (such as in Java or Ruby) or provided by 3rd party libraries.

Example (the $ sign is the command line prompt):

```
$ primecheck 32401
$ True

$ primecheck 3244568
$ False
```

Your second program will be named **primegen** and will take a single argument, a positive integer which represents a number of bits, and produces a prime number of that number of bits (bits not digits). You may not use the library functions that come with the language (such as in Java or Ruby) or provided by 3rd party libraries.

```
$ primegen 1024
$ 142405175064861448442669284843420489603593930617313976674095914073492903976984848373315014340583589674334422581561784146805278310143147937016874549483037286357105260324082207009125626858996989027
805604841776344359158053673248019204338406280932000275573354235703952211715047677821473373938293903583834167579544 3
```

```
$ primecheck 142405175064861448442669284843420489603593930617313976674095914073492903976984848373315014340583589674334422581561784146805278310143147937016874549483037286357105260324082207009125626858996989027805604841776344359158053673248019204338406280932000275573354237039522117150476778214733739382939035838341675795443
$ True
```

3.2 RSA Implementation (45%)

You will implement three programs: **keygen**, **encrypt** and **decrypt**.

keygen will generate a (public,private) key pair given two prime numbers.

Example (the $ sign is the command line prompt):

```
$ keygen 127 131
$ Public Key: (16637,11)
$ Private Key: (16637,14891)
```

In the table below you'll find some test cases for testing your **keygen** function. For those cases where you have to select your own numbers, make sure you select numbers that are at least 10 digits long.

First Number	Second Number	n	e	d
1019	1021	1040399	7	890023
1093	1097	1199021	5	478733
433	499	216067	5	172109
1061	1063			
1217	1201			
313	337			
419	463			

Student name:

encrypt will take a public key (n,e) and a single character c to encode, and returns the cyphertext corresponding to the plaintext, m.

Example:

```
$ encrypt 16637 11 20
$ 12046
```

In the table below you'll find some test cases for testing your **encrypt** function. For those cases where you have to select your own numbers, make sure you select n such that it is at least 20 digits long and that e is two digits long. The character c is a positive integer smaller than 256.

n	e	c	m
1040399	7	99	579196
1199021	5	70	871579
216067	5	89	23901
1127843	7	98	
1461617	7	113	
105481	5	105	
193997	5	85	
Student name:			

decrypt will take a private key (n,d) and the encrypted character and return the corresponding plaintext.

Example:

```
$ decrypt 16637 14891 12046
$ 20
```

In the table below you'll find some test cases for testing your **decrypt** function.

n	d	m	c
1040399	890023	16560	104
1199021	478733	901767	71
216067	172109	169487	101
1127843	964903	539710	
1461617	1250743	93069	
105481	41933	78579	
193997	154493	1583	

Student name:

3.3 Required Document (5%)

In addition to your source code and your output you are required to submit a memo to your instructor that is flawlessly written. No spelling errors. No grammatical errors, etc. If the document is deemed unprofessional, e.g. a significant number of grammatical or spelling errors, then it will be assigned a grade of zero.

If you're not sure about the format for a memo, then just search for "memo format" on Google.

Your memo should state clearly the status of the assignment. Is it done or not? Does it meet all of the requirements? If anything is missing then state clearly what is missing from your work.

Failure to give a status for the assignment will result in a grade of zero for the entire assignment. Providing a status that's false or significantly misleading will also result in a zero for the entire assignment.

Make sure your document is well-written, succinct, and easy to read. If you encountered problems with the assignment, then provide a detailed description of those problems and the solution(s) you found.

3.4 Assignment Submission (10%)

You are required to submit your work online, using the Blackboard. Late work will be accepted, however it is subject to late penalties as described in the syllabus.

Here are the requirements for submission:

- Submit your electronic copy using the Blackboard (attach the assignment as a compressed archive file (.zip, .tgz, .tbz2, .rar)
- The name of the compressed archive should be: *fistName-lastName*-PA-*assignmentNumber.zip* (e.g. Jane-Doe-PA-1.zip)
- Include (i) your memo, (ii) a completed handout with your results and your name on it, (iii) the source code, (iv) a README file that explains how to build and on how to run each program, (v) compiled executable(s) and (vi) a shell script or batch file that will compile your programs when executed; a Makefile (see GNU Make for details) would be great, however any form of script or building tool will do
- Include your e-mail address in the Comment field when submitting the assignment through the Digital Drop Box
- If for any reason you are submitting the assignment more than once, indicate this in the Comment field by including the word COMPLEMENT
