

ECE 543/444 Project 2: Implement RSA Algorithm

Introduction

In this project, you will first learn how to generate large prime numbers followed by implementing the RSA algorithm. 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 even long long int – 64 bits) of most computer languages will not suffice. Here are a few choices suggested:

Java has a built-in bigint class that behaves exactly like an int, except that it is not bounded by a finite value

C++ you will have to use a library such as NTL (<http://shoup.net/ntl/>) (Library for doing Number Theory) or GMP (<http://directory.fsf.org/wiki/GNU>) (the GNU Multiple Precision Arithmetic Library)

In Ruby you do not 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)

Students can use other coding languages than the suggested ones above.

This project 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.

Part 1: Prime Numbers (50%)

RSA requires finding large prime numbers very quickly. **You need to implement a method for primality testing of large numbers. You are suggested to implement the Miller-Rabin test as studied in ECE543/444.** It is also allowed to implement other well adopted test methods such as Solovay–Strassen primality test or Frobenius pseudoprimalty test. Your task is to develop two programs (make sure you use your two programs to check each other).

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 the 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
$ 14240517506486144844266928484342048960359393061731397667409591407
34929039769848483733150143405835896743344225815617841468052783101
43147937016874549483037286357105260324082207009125626858996989027
80560484177634435915805367324801920433840628093200027557335423703
9522117150476778214733739382939035838341675795443

$ primecheck 14240517506486144844266928484342048960359393061731397
66740959140734929039769848483733150143405835896743344225815617841
46805278310143147937016874549483037286357105260324082207009125626
85899698902780560484177634435915805367324801920433840628093200027
5573354237039522117150476778214733739382939035838341675795443
$ True
```

Part 2: RSA Implementation (50%)

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			

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	

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	

Project Report and Code Check

You need to write a technical report for this project. The report should describe the algorithms that are applied in your codes. You should have executable codes to support your results. Using your codes, we should be able to generate the results you present in your report.

- Your report is **due at 23:59 pm, April 30, 2023, Chicago time**. Submit your technical report through the blackboard system with document named as **"ID_Lastname_prjt2_report.pdf"**. Please attach your source codes to your report.
- Your codes need to contain executable files for **primecheck**, **primegen**, **keygen**, **encrypt**, and **decrypt**. TA will examine both your source codes and the usability of your executable files.
- **Late submission will not be graded and lead to a "0" grade of this project.**
- You need to independently work on the project. **Projects identified with plagiarism problem will get a "0" grade.**