COSC 483/583: Applied Cryptography

Programming Assignment 2: RSA **Due: 11:59:59 pm November 1, 2019**

Ground Rules. You may choose to work with up to two other student if you wish. Only one submission is required per group, please ensure that both group members names are on the submitted copy. Work must be submitted electronically via `git`. **You are expected to re-register your group using the same method as programming assignment 1.** You do NOT need to re-submit your crypto keys. The choice of programming language is yours, but your software will be expected to operate in an environment of my choosing, specifically an arch linux virtual machine.
I will expect all listed software artifacts (executable programs) to exist in the top level of your project directory post execution of the `make` command.
In addition to the executables listed below, you are expected to provide the following deliverables, any missing deliverables will result in point loss:

- Source code for your system, file organization is up to you.
- A Makefile at the top level of the repo named `Makefile` which will result in the appropriate software artifacts being generated (or a blank Makefile if compilation is not needed).
- A file named groupMembers.txt at the top level of your repo containing all group members, this *is* required for groups of size one.

Task 1) Implement RSA Key Gen Your first task is to correctly implement RSA keygen. You are NOT allowed to use built in implementations of RSA key generation.
Key File Formats Key files should be three lines long, containing Strings representing the integer components of the key values in base 10. For public key files the first line should contain the number representing the number of bits in N, the second line should contain N, and the third line contains the number representing e. For the private key files, the first line should contain the number of bits in N, the second line contains N, and the third line contains the number representing d. Example files will be posted on the website.
rsa-keygen should take the following argument flags:

- -p `<public key file>`: required, specifies the file to store the public key
- -s `<secret key file>`: required, specifies the file to store the private key
- -n `<number of bits>`: required, specifies the number of bits in your p and q

Task 2) Implementing Padded RSA Your next task for this assignment is to correctly implement an asymmetric key cipher. As expected, you are not allowed to simply use a library's implementation of RSA. You can You are expected to implement RSA key generation and padded RSA encryption/decryption (Construction 11.30 in the text book). assume that the size of r in your implementation should be half of n, you may assume that all inputs will fit appropriately in a single encryption element.

- rsa-enc : encrypts an integer using RSA
- rsa-dec : decrypts an encryption of an integer using RSA
- rsa-keygen : creates a valid RSA public key/private key pair and stores them in files

rsa-enc and rsa-dec should take the following argument flags:

- -k `<key file>` : required, specifies a file storing a valid RSA key in the example format
- -i `<input file>` : required, specifies the path of the file containing an integer in Z_n^* in String form (base 10) that is being operated on
- -o `<output file>` : required, specifies the path of the file where the resulting output is stored in String form (base 10)

Extra Credit Portion There is one piece of extra credit that will be awarded if you use your own implementations of a specific function rather than a provided implementation from a library. If you attempted the extra credit, please clearly mark in a file named README that you did so.

- Implement generation of random prime numbers of a specific bit length. Your testing of the primality of a potential candidate number should use a probabilistic prime test, you should implement the test based on research of the appropriate algorithms. A recommendation is the Miller-Rabin primaility test.