CS378, Spring 2022
Homework 6
due: April 15

Topic: Implementing RSA Cryptosystem Programming Project

1 Overview

The objective of this project is to implement components of the RSA cryptosystem. You will implement three modules that make up this cryptosystem.

Key setup: This module will compute and output the keys: public and private. The keys will be output to two separate files named *public_key* and *private_key*. The *public_key* file will consist of two integers, n and e, each given in a separate line.

Encryption: This module will take the public key and a message to be encrypted as the inputs. They will be read from the files *public_key* and *message*, respectively. The module will output the ciphertext (encrypted message) which will be stored in a file named *ciphertext*.

Decryption: This module will take the public key, the private key and the ciphertext to be decrypted as the inputs. They will be read from the files *public_key*, *private_key* and *ciphertext*, respectively. The module will output the decrypted message and store it in a file named *decrypted_message*.

In the project you will have to do computations on very large integers (at least 200-digit numbers in decimal). Therefore, you will need do use a language with built in large integers (such as Python) or use existing large integer package (such as PARI/GP). You can also choose to implement your own large integer arithmetic package.

2 Key setup

To compute the keys you need to implement a modular exponentiation algorithm, that is, an algorithm that takes integers x, a and n, and returns $x^a \pmod{n}$. Use the algorithm described in class. Modular exponentiation will also be used in encryption and decryption modules.

You will need to implement an algorithm to generate large prime numbers. Two key components of that algorithm are: (1) an algorithm to generate large integers at random, and (2) a primality testing algorithm. You can use Fermat Primality Test or Miller-Rabin Primality Test.

Finally, you will have to implement the Extended Euclid Algorithm, needed to compute multiplicative inverses.

To set up a public key you will generate two different prime numbers p and q with at least 100 decimal digits each, and making sure the difference between them is at least 10^{95}.

Next you will compute $n = pq$ and choose an integer e relatively prime to $(p-1)(q-1)$. Usually, $e = 2^{16} + 1 = 65537$ is a good choice (you will need to check that it works). At this point your public key (n, e) is ready. To compute the private key d you will apply the Extended Euclid Algorithm to compute $d = e^{-1} \pmod{(p-1)(q-1)}$.

3 Encryption and Decryption

The message to be encrypted will appear as a sequence of digits (an integer with al least 150 decimal digits but within the bound given by n). You will encrypt it using the public key and the modular exponentiation algorithm

To decrypt the ciphertext, you will use the private key and apply the modular exponentiation algorithm.

4 Programming languages/systems

To write your program you have to use one of the following programming languages: Python, C/C++, or Java. Python and Java have large-number arithmetic built in. Examples of C libraries implementing large-number arithmetic are: PARI/GP (available at http://pari.math.u-bordeaux.fr/) and GMP (available at http://gmplib.org/).

You cannot use source code for any components of this project from the web or elsewhere.

If you use C/C++, you should use OpenStack ubuntu virtual machine (the common choice in other CS classes) to write and compile the source code. If you use Java or Python, use Java 1.8, python 2.7.12 or python 3.5.2. If you have questions contact the grader directly.

5 Documentation

Submit a single .zip file which should contain the following files:

- A narrative report (in pdf format) describing how your program works and what algorithms you implemented. List the major components of the program and explain how they fit together. Describe how you tested the program and how you verified its correctness. Write about the problems that you encountered and how you overcame them. If your program does not work correctly, explain what parts do not work and what parts you believe work correctly. The report should also contain compilation and execution instructions for your program.

- The source code for all modules implemented, thoroughly documented. In each module explain the functionality of the module (what is the input and the output). Describe in comments all important variables.
- The results of a test run for the message (a single integer of 180 digits):

  777777777777777000000000000000222222222222222333333333333333444444444444444
  222222222222222555555555555555666666666666666777777777777777888888888888888
  999999999999999000000000000000

  as plaintext. The results of the test should be documented by including the files *public_key*, *private_key*, *message*, *ciphertext* and *decrypted_message* as described in the Overview Section.