RSA Public Key Generation Project: Design Document

Sam Demorest and Jason Pearson

CS 6310

Dr. Dedoncker

**Design Overview**

Our program is made up of two separate programs. One program is for RSA key creation, and the other program will implement asymmetric encryption using the RSA algorithm and the keys created by the first program. Generating RSA key pairs requires the generation of large prime numbers. These functions will be split off into a separate library that deals with primality tests and other things related to prime numbers. The idea is that this primes library will be reusable for other projects. The reason for this is so other programmers can find our prime number generation code and use it in their projects as well.

The prime generator will use a two different primality tests. To generate relatively small (but still large) prime numbers, we will use the AKS Primality Test, which is the best performing deterministic primality test currently known. This is more to demonstrate an understanding of the principles behind the AKS test rather than to implement it for any serious purpose in our key generator. The AKS algorithm is still too slow for extremely large primes, so in the actual key generation process, we will use a series of Miller Rabin probabilistic primality tests to generate a number that can be determined to be prime within a probability threshold. We will be using various optimizations to the probability tests, such as not considering even numbers, in order to increase the speed at which we are able to determine whether or not a number is prime.

The key generator will call the prime generation method of the first program and that will take care of all the work for prime number generation. After it gets the prime numbers p and q we use those to compute n which is simply p times q. Next we need to determine φ(*n*) which is simply *n* - (*p* + *q* -1). After these are all computed we create a good e value and compute d. The e value is simply a number 1 < e <φ(*n*). Then d = e-1(mod(φ(*n*)), which is the private key part of the encryption.

This project will be completed in the Rust language, a language which is being developed by the Mozilla Foundation. Rust is meant to be a memory-safe C++ replacement. It focuses on memory safety by enforcing strict rules at compile time. Similar to Ada, if a Rust program compiles, it is very unlikely that it will run into runtime errors.

The modularization of the programs we are writing should help to ensure that we are able to achieve maximum code reusability for future projects, and we may even be able to produce a useful library for primality testing on BigInt data types. As the Rust language is still under very active development (version 1.0 Beta has yet to be released), we may freeze the versions of some libraries we use as to not lose compatibility in our code with APIs that are still unstable.

The project workflow uses Github as a collaborative development environment. This works nicely with Rust, as the Rust build manager, Cargo, initializes a git repository every time a project is created. Rust is designed with modern development techniques in mind, including git integration and built-in unit testing functionality. We will be utilizing these amenities as much as possible in our development cycle.

**AKS References:**

http://www.cse.iitk.ac.in/users/manindra/algebra/primality\_v6.pdf

http://mathworld.wolfram.com/AKSPrimalityTest.html

**Rust References:**

https://doc.rust-lang.org/

http://rustbyexample.com/

https://github.com/rust-lang/rust

**RSA Cryptosystem References:**

http://mathworld.wolfram.com/RSAEncryption.html

http://stackoverflow.com/questions/12749858/rsa-public-key-format

https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture12.pdf