CS 308: Networking and Distributed Systems  
Fall Term, 2011

Lab 3

In this lab, you'll be implementing RSA and some of the supporting number theoretical algorithms.

Implementing RSA

Implement a Java program that illustrates the RSA algorithm. Specifically, your program should find 2 primes, use them to generate a public/private key pair, ask the user for a message (i.e. a number), encrypt this value, and then decrypt the result. Print out each stage of this process and make it possible to set different desired bit lengths for the primes. Use the java.math.BigInteger class for your numbers; it is a class that represents arbitrary precision integers. Conveniently, it also has functions for identifying “probable primes” of a given size and computing the multiplicative inverse of a prime mod n (the function is called modInverse). The Scanner class also provides a nextBigInteger method to read them in.

After completing the implementation of RSA, you will spend the rest of the period examining aspects of the implementation. Begin by comparing the running time of at least two different approaches to encryption/decryption. The naive way to do this is to use pow to raise the message to the necessary power and then mod to get the ciphertext. Better is to use modPow, which does both at once. Compare the times of these approaches as the bit lengths increase. (java.util.Timer will be handy for this.)

Next, try to start implementing the parts yourself. Begin by implementing Euclid’s algorithm to find greatest common divisors. Then extend this to also report the coefficients to the linear combination of the input values that give the gcd. (This takes a bit of care, but can be done by using the recursive values.) The result can be used to replace the call to modInverse.

Finally, work on using repeated squaring (as discussed at the very end of class on Wednesday) to replace modPow.