
Problem Set 2

This problem set is due on *Monday, February 25, 2008* by 8AM. You must submit your homework to your edlab CS591d course directory. Make sure that your homework files have proper permissions set such that the course staff can read the files, but not other students. Do not make any modifications to your homework files after the due date, unless you notify the TA that you are using a late pass. You may write your solution in any program (Word, L^AT_EX, etc.), but you must convert your document to PDF for submission. We will accept only PDFs. Please contact the TA well ahead of the deadline if you have a question about these procedures.

This is an individual problem set. See Handout 1 (*Course Information*) for our policy on collaboration and late penalties. Mark the top of each page with your name, cs591d, the problem set number and question, and the date. Each solution must begin on a new page. Points may be deducted if your TA has problems understanding your solution. In mathematical problems, **show all your work**.

Each team member should turn in the individual problem on his or her own. For the group problems, have your team submit a *single solution*.

Problem 2-1. Inverses and Modular Exponentiation (3 pts)

This is an individual problem. As customary, if you receive any key insights from someone else or some other resource, you must cite that person or resource.

Compute the following inverses using the Euler extension to FLT. Exponentiation should be done with square and multiply. All steps of exponentiation need to be shown. Show all your work and verify correctness by showing that x multiplied by its inverse = 1.

1. Solve for x . $x = 1/13 \pmod{73}$
2. Solve for x . $x = 1/18 \pmod{133}$
3. Solve for x . $x = 1/17 \pmod{143}$

Problem 2-2. To CRT or not to CRT, that is the question (6 pts)

This is an individual problem. As customary, if you receive any key insights from someone else or some other resource, you must cite that person or resource.

This problem involves a large amount of programming and testing. We highly recommend you start early on this assignment.

There are two common ways to implement the exponentiation step for RSA decryption. You could simply compute $m = c^d \pmod{n}$. But the Chinese Remainder Theorem can speed up the process. This problem will have you explore the effects of the CRT on the performance of RSA decryption.

We recommend you use the Java `BigInteger` package for multi-precision numbers (32-bit ints won't live up to your expectations!). Most programming languages have a library that provides an

abstraction for multi-precision (large number) arithmetic. The GNU MP Bignum Library is a very sophisticated library for C++, for instance. You may choose whatever language is most convenient.

Because we have not yet covered primality testing in lecture, you are free to use any built-in procedures for primality testing or prime number generation. Ask the TA if you have any questions about what tools you are permitted to use. You can (and should) assume you have a source of randomness. You may not borrow any code for cryptography beyond these large number packages.

(a) Implement RSA with and without CRT (2 pts)

Implement RSA with the modulus as a product of two primes of equal length (see Stinson pages 173–175). You will need to write at least three functions: `generateKey`, `encrypt`, and `decrypt`. You may use the built-in modular exponentiation function, if it exists. Otherwise you should implement the square-and-multiply algorithm. `generateKey` should take as input an integer k representing a bit length, then output all the RSA parameters (p, q, n, e, d) where $n = pq$ and the bit length of each prime = k . That is, $|p| = |q| = k$. Note that Stinson uses notation (b, a) when we use notation (e, d) . Make sure the MSB of each prime is a 1, otherwise your performance results may be skewed. `encrypt` should take a numeric message m (e.g., `BigInteger`) and (e, n) to produce a ciphertext $c = m^e \bmod n$. `decrypt` should take a ciphertext and (d, p, q, n) to produce the original message. You will implement `decrypt` in two ways: one with CRT and one without CRT. Submit your code and demonstration that encryption and decryption actually work.

(a) Prime time (4 pts)

Graph the performance of RSA decryption with and without the CRT for various key bit lengths. Because decryption could run well in the sub-millisecond neighborhood, you should record the time it takes to do a large number of sequential decryptions for a given public key pair (e.g., 100 decryptions). Measurement of a single decryption might not produce a meaningful or representative result.

We suggest starting with 64-bit primes, then work your way up to 1024-bit primes in 64-bit increments. Do a side-by-side comparison in graph format (x-axis = bit length of each prime, y-axis = running time of the decryption). At what bit length do you find that CRT performs much better than naive modular exponentiation? Does the naive approach sometimes perform better than the CRT? Why? Submit your answers and a properly labeled graph.

Note that we are expecting a large set of different answers. Different implementations, different programming languages, and different computers will influence performance. If you enjoy designing experiments that have rigorous methodology for measurement, you might enjoy graduate school!

Problem 2-3. Term project (1 pt)

This is a group problem. Exactly one team member should submit this problem with all team member names listed.

Give a 2–3 paragraph summary of the term project idea your team is considering. If you haven't narrowed down the idea to a single topic, then list at most two potential topics. The summary should cover the problem you will investigate along with a paragraph discussing related work from existing academic publications. scholar.google.com is a good start. Talk to your favorite librarian for help.