

HW # 9

Due Date: Tuesday, November 28th, 2007, 7:20 PM
at the beginning of class

Reading Assignment:

- W. Stallings, *Cryptography and Network Security*, Chapter 11
- A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography* Sections: 9.1-9.3, 9.4.2, 9.5.1-9.5.2

Problems:

1. Compute the following exponentiations $x^e \bmod m$ applying the square-and-multiply algorithm (also called left-to-right binary exponentiation):
 - (a) $x = 5, e = 83, m = 101$
 - (b) $x = 7, e = 199, m = 101$After every iteration step, show the exponent of the intermediate result in binary notation.
2. Write a C program which computes $x^e \bmod n$ using the square-and-multiply algorithm (also called left-to-right binary exponentiation). Two tips:
 - Use `long` type variables for all operations.
 - Make sure you apply the modulo reduction in *every* step of the algorithm. Otherwise you run very rapidly out of the 32 bit range covered by long type variables.
 - (a) Provide a print out of the C program as well as a sample run.
 - (b) Use your program to verify your results from Problem 1.

3. As we saw in the lecture, the modulus of RSA has been enlarged over the years in order to thwart the dramatically improved attacks. As one would assume, public-key algorithms are becoming slower as the word length increases. We will study the relation between word length and performance in this problem. The performance of RSA, and of almost any other public-key algorithm, is dependent on how fast modulo exponentiation with large numbers can be performed.

Assume that one modulo multiplication or squaring with k bit numbers takes $c \cdot k^2$ clock cycles, where c is a constant. How much slower is RSA encryption/decryption with 1024 bits compared to RSA with 512 on average. Only consider the encryption/decryption itself with an exponent of full length and the square-and-multiply algorithm.

4. There are ways to improve the square-and-multiply algorithm, that is, to reduce the number of operations required. Although the number of squaring is fixed, the number of multiplications can be reduced. Your task is to come up with a modified version of the square-and-multiply algorithm (also called left-to-right binary exponentiation) which requires fewer multiplications. Give a detailed description of how the new algorithm works and what the complexity is (number of operations).

Hint: Try to develop a generalization of the square-and-multiply algorithm which processes more than one bit at a time. The basic idea is to handle k (e.g., $k = 3$) exponent bits per iteration rather than one bit in the original square-and-multiply algorithm.

5. In this problem we are examining the effect of the Chinese Remainder Theorem on RSA decryption. Assume we have RSA with the following parameters: $p = 11$, $q = 13$, and $d = 113$.
 - (a) Decrypt the message $y = 40$ as $x = y^d \bmod n$ on paper using square-and-multiply algorithm. Show every single step. For multiplication and squaring you can use any method you like and it is sufficient to show just the result of the multiplication and squaring operations.
 - (b) How many multiplications and how many squaring operations did you have to perform, how many total?
 - (c) Recompute y using the Chinese Remainder Theorem.
 - (d) How many multiplications, how many squarings and how many operations in total did you have to perform now?
6. Use a spreadsheet (such as Excel), or a calculator, and test all odd numbers in the range from 219 to 233 for primality using the Miller-Rabin test with base 2. Document results of all intermediate modular multiplications. Determine the number of modular multiplications.

Extra Credit Problems:

The extra credit problems are due December 5th at the beginning of class. You can receive up to 2 points extra credit toward the final grade. This is approximately equivalent to 8 extra points in the midterm.

1. Use the Chinese Remainder Theorem to derive general formulas for the 9 messages that are not concealed using RSA encryption (i.e., the ciphertext value is identical as the message value). Demonstrate the correctness of your formulas using at least two examples.
2. Write a C program that computes $x = a \cdot b$ for integers of size 512-bit using the paper and pencil algorithm for multiplication. Do **not** use any long number, multiprecision library. Provide a print out of the C program as well as a sample run.
3. Write a C program that tests all numbers in the range $2^{31} < p < 2^{32}$ for primality. Hint: For this problem you can use one of the multiprecision libraries listed here.
 - (a) Determine the amount of composite numbers eliminated using fast test division by all primes smaller than $B=1000$.
 - (b) Compare this number with the theoretical predictions based on the Merten's theorem.
 - (c) Determine the amount of composite numbers that were not eliminated by division by small primes but were eliminated using Miller-Rabin test with base 2.
 - (d) Determine the amount of composite numbers which passed more than a single iteration of the Miller-Rabin test.
 - (e) Determine the total amount of prime numbers in the given range. Compare this number with the theoretical predictions based on the value of the function π .
 - (f) Determine the average distance between two consecutive primes in this range. Compare this number with the theoretical predictions based on the value of the function π .
 - (g) Determine the minimum and maximum distance between two consecutive primes in this range. Provide values of primes separated by the minimum and the maximum distance.
 - (h) Provide a print out of the C program as well as a sample run.