Institut for Matematik og Datalogi
Syddansk Universitet

October 30, 2003
JFB

Introduction to Computer Science
E03 – Lecture 9

Lecture, October 27

We covered security from section 3.7 in the textbook and began on cryptology (not following the textbook, except a bit from section 11.6; there are some notes on cryptography, from PGP, using a link on the course's homepage).

Lecture, November 3

We will continue with RSA and exponentiation and then cover the first four or five sections of chapter 11.

Lecture, November 10

Kim Skak Larsen will be lecturing on chapter 9.

Discussion section: week 45 – Terminal Room

Discuss the following problems in groups of two or three. You will be using the program `xmaple`.

The best known public key cryptographic system, RSA, was presented in lectures. It is one of the systems included in PGP and GPG. Its security is based on the assumption that factoring large integers is hard. (The system you are using in PGP is based on discrete logarithms, rather than factoring, but the problems are similar in many ways. The factoring is easier to understand and test in Maple.)

A user's public key consists of a large integer n (currently numbers with at least 1024 bits are recommended) and an exponent e. The integer n should be a product of two prime numbers p and q, both of which should be about half has long as n. Thus, in order to implement the system it must be possible to find two large primes and multiply them together in a reasonable amount of time. For the security of the system, it must be the case that no one who does not know p or q could factor n.

At first glance this seems strange, that one should be able to determine if a number is prime or not, but not be able to factor it. However, there are algorithms for testing primality, which can discover that a number is composite (not prime) without finding any of its factors. (The ones most commonly used are probabilistic, so they could with small probability declare a composite number prime; the probability of this happening can be made arbitrarily small.)

Using Maple, you should try producing primes and composites and try factoring.

1. Small numbers.

   Start your Maple program. Type `restart;` at the beginning to make it easier to execute your worksheet after you have made changes. You can do this from `Execute` in the `Edit` menu.

   Use *help* to find out about the function *ithprime*. Experiment to find out approximately how big a prime it can find. When it cannot find such a big prime, you can use the **STOP** button in order to continue (it is a hand in a red background). To assign a value to a variable, you use the assignment operator `:=`; for example `x:= ithprime(4);`. Multiply two of the large primes it finds together, and try to factor the result, using the function *ifactor*. Notice how quickly the factors are found for these small numbers. (Large numbers are clearly necessary for security.)

2. Finding larger primes.

   In order to find good prime factors p and q for use in RSA, one can choose random numbers of the required length and check each one for primality until finding a prime.

   Maple contains a function *isprime* which will test for primality. Try it on some some small numbers, such as 3, 4, 7, 10. Maple has another function *rand* which returns a random 12-digit number. Try typing `x:=rand();` and check if your result is prime. Rather than executing these commands until you find a prime, you can use a *while loop*. You want to continue creating new random numbers until you get a prime, so you can type `while (not isprime(x)) do`, ignore the warning, and type `x:=rand();` on one line and `end do;` on the next. How many different values were chosen before a prime was found? (I got 33, but you could get another number.) Now create a second prime called y (remember that y will need some value before your start your *while loop*). Multiply x and y together and try factoring the result. This should also go relatively quickly.

3. Finding even larger primes.

   To get random numbers which are twice as long, you can create two random numbers a and b and create $10^{12} * a + b$. Unfortunately, two calls to *rand* in the same statement will give the same result both times, so you need to choose values for a and b independently and then combine them. (Suppose you typed `m1 := 10^12*rand() + rand();`. Why wouldn't you ever find a prime testing values found this way?) Try finding two primes, each 24 digits long. The first can be found by starting with `m1:=4;` so you start out with a composite. Then use the following: `while (not isprime(m1)) do`, `a := rand();`, `b := rand();`, `m1 := 10^12 * a + b;`, and `end do;`. Multiply the two primes together and try to factor the result. Use the **STOP** button on the toolbar after a few minutes; the computation takes too long. As you might imagine, no known algorithm would factor a 1024-bit (about 300 digits) number on your PC in your lifetime. It is easy to find the primes and multiply them together, but it is very difficult to factor the result! (Or RSA would not be secure.)

   To get a feeling for how long it takes to factor numbers of different lengths, try changing the 10^{12} in your *while loops* to 10^9 and 10^10. With 10^9, it will probably take less than two minutes, and with 10^7, probably about three minutes.

4. Discuss questions 2 and 3, and 5 on page 461.

5. Discuss question 2 on page 489.

6. Discuss questions 20 and 25 on page 491.

Assignment due 8:15, November 11

Late assignments will not be accepted. Working together is not allowed. (You may write this either in English or Danish, but write clearly if you do it by hand.)

1. Question 4a on page 490.

2. Question 14 on page 490.