Institut for Matematik og Datalogi
Syddansk Universitet

November 5, 2004
JFB

Introduction to Computer Science
E04 – Lecture 9

Lecture, November 1

Rolf Fagerberg lectured on file structures from section 9.5. Then we continued with security.

Lecture, November 8

We will begin on encryption from section 11.6 and discuss exponentiation, an efficiency concern with RSA and many other public key cryptosystems. There are some notes on cryptography, from PGP, using a link on the course's homepage.

Lecture, November 15

We will begin on the theory of computation from chapter 11.

Supplementary notes on RSA

The textbook leaves out many important details regarding the implementation of RSA. For example, it gives the incorrect impression that in computing $m^e \pmod n$ that one would first compute m^e and then reduce modulo n. This is not what occurs in practice since it is infeasible for the large numbers used. The intermediate result would have about $e \log(m)$ bits, which would usually be more than 2^{500} bits (either m^e or c^d would be extremely long)! Thus, one computes this using intermediate computations and reducing modulo n after each step. This works because of the following:

Lemma. For all nonnegative integers a, b and any integer $n > 1$,
$a \cdot b \pmod n = (a \pmod n)(b \pmod n) \pmod n$.

Note that this can be proven using the fact that $a = x \pmod{n}$ if and only if $0 \leq a < n$ and there is an integer k such that $a = x + k \cdot n$.

The powers can be computed efficiently using the following algorithm:

```
function power(a,exp,n)

# Compute a^exp (mod n) for nonnegative exp

   if exp = 0 then return(1)
   else if (exp is odd) then
           return((a*power(a,exp-1,n)) mod n)
        else
           c <- (power(a,exp/2,n))
           return((c * c) mod n)
```

The values e and d are *multiplicative inverses* of each other modulo $(p-1)(q-1)$ (i.e. $e \cdot d \pmod{(p-1)(q-1)} = 1$). They can be computed by using the Extended Euclidean Algorithm, which computes greatest common divisors.

Def. $\gcd(a,b)$ = greatest common divisor of a and b = largest $d \in \mathbb{Z}$ (the integers) such that $d|a$ and $d|b$

If $\gcd(a,b) = 1$, then a and b are *relatively prime.*

Thm. $a, b \in \mathbb{N}$ (nonnegative integers). There exist $s, t \in \mathbb{Z}$ such that $sa + tb = \gcd(a,b)$.

Claim: The integers $d = gcd(a,b)$, s and t can be found efficiently, using the Extended Euclidean Algorithm.

For RSA, the value e is chosen so that $\gcd(e,(p-1)(q-1)) = 1$. To find d, we also need a value k such that $e \cdot d = 1 + k(p-1)(q-1)$. Thus, we can compute d by solving for s in the equation $se + t(p-1)(q-1) = 1$. This can be done using the Extended Euclidean algorithm since $gcd(e,(p-1)(q-1)) = 1$.

Discussion section: week 46 – Terminal Room

Discuss the following problems in groups of two or three.

First, you will be using the program `gpg` to try encryption. Usage information can be obtained by typing `gpg -h | more` (hitting the space bar will get the rest of it; the vertical line says pipe the output through the next program, and `more` shows a page at a time).

1. Create a public and private key using `gpg --gen-key`. You should choose DSA and El Gamal, and size 1024. Go to the directory `.gnupg` using `cd .gnupg`. List what is in the directory using `ls -al`. Try the commands `gpg --list-keys` and `gpg --fingerprint` to list the keys you have, with the fingerprints, which make it easier for you to check that you have the correct key from someone. How would you use fingerprints?

2. You can save your public key in a file in a form that can seen on a screen using `gpg --export -a Your Name >filename`. You are "exporting" your key and specifying where the output should go. Then look at it using `more filename`; the `-a` made it possible to see it reasonably on your screen, since it changes it to ASCII.

3. Mail this file to someone else. (Either another group or within your own group.)

4. Try to figure out how to use `gpg` to "import" the public key you got from someone else. Check the fingerprint.

5. Create a little file and encrypt it. You can use `gpg -sea filename`. What does this do?

6. Mail your file to whoever has your public key. Read their file using the command `gpg -d inputfile >outputfile`. Then look at the output file you created.

7. You can also encrypt a file for your own use using a symmetric key system protected by a pass phrase. Try using `gpg --force-mdc -ca filename`. Then try decrypting as with the file you decrypted previously. Why might you want to do this?

In the following, 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 GPG 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 $10 * *12 * a + b$ (10 raised to the power 12 times a plus 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;
end do;.
```

   Multiply the two primes together and try to factor the result. If your machine is not fast enough, use the **STOP** button on the toolbar after a few minutes; the computation takes too long. Otherwise, use one of the primes you already have and create another with 36 digits, multiply them together and try factoring them. 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.)

Try to get a feeling for how long it takes to factor numbers of different lengths; you can try changing the $10 * *12$ or $10 * *24$ in your *while loops* to larger or smaller values.

4. Find the multiplicative inverse of 25 modulo 43. You could try using `xmaple` and finding out about the function for computing the Extended Euclidean Algorithm by typing `?igcdex`. Does it help?

5. Discuss questions 2 and 3, and 4 on page 489.

6. Discuss problems 50 and 52 on page 493.

7. Discuss issues 2, 6 and 7 on pages 493–494.

Assignment due 8:15, November 16

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.) Show your work where it is relevant.

1. Find the multiplicative inverse of 39 modulo 77.

2. Find four different square roots of 4 modulo 77 (numbers which multiplied by themselves modulo 77 give 4). Note that all of these numbers should be less than 77.

3. Add two of these different square roots which are not negatives of each other modulo 77 (two where adding them together does not give 77). Find the greatest common divisor of this result and 77. Subtract these same two different square roots and find the greatest common divisor of this result and 77. Think about why you get these results.

Announcement

Der er Matalogifest lørdag den 13. november. Festen afholdes i U26. Temaet er "Jubilæumsfest" i anledning af IMADA's 100000 års jubilæum og Matalogifestens 10000 års jubilæum. Tilmelding kan foretages på sekretariatet. Pris: 125 kroner.