Institut for Matematik og Datalogi
Syddansk Universitet

December 6, 2001
JFB

Introduction to Information Technology
E01 – Note 8

Lecture, November 30

We finished section 3.6 and briefly touched on all the topics (social issues) in chapter 14. Classical and public key cryptography were introduced. Sorting from the algorithms lab was discussed, including the algorithm Quick Sort.

Lecture, December 14

We will continue with cryptography: classical and public key cryptography, RSA, and digital signatures. Encryption and digital signatures with PGP will be demonstrated.

Announcement

If you use up your 250 printed pages in any semester, you can buy extra printing credits, by contacting

Liselotte Nielsen
Center for Interaktive Medier

Her direct phone number is 3615. The price is 100kr. for 250 credits (where one credit equals one printing page).

Primary Lab 8 - for week 51

The purpose of this lab is to introduce cryptography, both from an applications point of view and from an algorithmic point of view. PGP will be used, as will Maple.

You can find a short introduction to cryptography and PGP through the WorldWideWeb. From the Web page for this course, you can find a link. The most relevant sections are "What is cryptography?", "Public key cryptography", "How PGP Works", "Keys", "Digital signatures", and "What is a passphrase?". If anyone ever asks you to certify his/her certificate, you should read the appropriate parts of that article before you do it.

You get information on how to use PGP similarly. You can start reading this, and read the relevant section in the introduction to cryptography when there is something you do

not understand. After getting the page **PGP 7.0 Windows 95/98/NT/2000 User's Guide**, click on **PDF format** (a standard format for documents). In the Bookmarks, click on **Table of Contents** and there click on chapter 3 **Making and Exchanging Keys**. This chapter, along with chapters 5 and 6 tell about creating keys, encrypting e-mail, and encrypting files.

Exercise 1

Making your keys.

First you will need to specify where your keys should be stored. To do this, click on the **PGPtray** icon (which looks like a lock) in the System tray, which is in the lower right hand corner of your screen. (You should also be able to get it from the **Start** button in the lower left-hand corner.) From the menu, choose **Options...→ Files**. In the field **Public keyring file**, remove everything that appears from **C:** through **PGPNT** and replace it with **H:**. You can do this by selecting the part you want to delete and then typing `H:`. Do the same for the field **Private keyring file**. Click on **OK**.

Now, you are ready to create your keys. From the **PGPtray**, choose **PGPkeys**. You may get a warning message. If you do, in order to continue with this assignment, click on **Don't warn me again** and **Continue**.

After reading the first message, click on **Næste**. When it asks for your name and e-mail address, fill them in, and click on **Næste**.

From this point on, keep clicking on **Næste** until you can click on **Udfør**. All the defaults are fine. The only time you need to do anything different is when it asks for a **Passphrase**. There you should choose a good secret password or phrase which is long enough.

After your key appears in the window which is open, you can exit that window. You will be asked to save your keys, but if you do not have a diskette with you, you will have to click on **Don't save** and save the keys at some later time (assuming you expect to use them again).

Exercise 2 – Optional. Not hard. Do if you have time.

Encrypting and decrypting files.

From **PGPtray**, choose **PGPtools**. A small set of buttons will appear. The second to left button (an envelope with a lock) is for encrypting files. Click on it. Click on the arrow next to the **Søg i** with **PGPNT**. Find **Dokumenter** and click on it. From there select the file you wish to encrypt (probably a .txt file). From the **Key Selection Dialog** window, choose your own key (double click on it so it appears in the **Recipients** area). Click on **OK**. The file should get encrypted now.

Go into **Dokumenter** and find the file you encrypted. Notice that there is an encrypted version too, now. (Assuming you were doing this for security, you would delete the original now.) When you double click on the encrypted version, you can't read it. Click on **OK**.

The fifth button in **PGPtools** is for decrypting. Click on it. Now select your file, as it asks, and click on **Åbn**. You need to enter your passphrase when it asks. Change the name of the output file. Check that the output file is the same as your original one. Now delete one of the two identical files.

Exercise 3

Sending your public key to someone.

Agree with someone sitting near you that you will send him/her your public key. Get their e-mail address. Open PGPkeys again. Select your key pair and then click on **Copy** from the **Edit** menu. Open *Outlook Express*, click on the white icon to the left for creating a new message, click on the part of the window where the message should go, and click on **Sæt ind** from the **Rediger** menu (or **Paste** from your **Edit** menu). Now fill out the *To* and *Subject* fields to send your message to your neighbor. Finally, sign your message as follows: Click on the toolbar icon next to the keys, the one which looks as if someone is signing a document (make sure it says **Sign (PGP)** or **Sign(P...)**, and do not worry that nothing seems to happen), click on **Send**, and then type in your passphrase. Send your public key to your lab instructor, too.

Exercise 4

Verifying signatures.

Read the e-mail from your neighbor and the one from your instructor (you should double click on the message to get a new window for it). To verify the signature, you can click on the icon next to the keys which says *Decrypt PGP message*. When you are asked to select the key would would like to import to your keyring, click on **Import**. (If this does not appear, either you already have the relevant key in your keyring, or you need to click on *Decrypt PGP message* again, because it only checked the signature the first time.) This should have put the the public key on your public keyring.

To check that nothing happened with the e-mail while it was being sent, you can verify the public key by checking the fingerprint. To do that, open *PGPkeys*, select the key you want to verify, choose **Properties** from the **Keys** menu, and compare the *Fingerprint* to what your neighbor has a fingerprint for his/her own key. Assuming they are the same, you can trust the key as belonging to that person.

Exercise 5

Encrypting and signing.

Now that you have your neighbor's and instructor's public keys, you can encrypt mail that you send to them. Follow the directions starting on page 97 of the *User's Guide*; perform all 7 steps. (Use the PGP encryption button, not the other encryption button.)

Exercise 6 – Optional. Not hard. Do if you have time.

Reading and verifying encrypted e-mail.

Try reading the encrypted e-mail your neighbor sent you. You will have to decrypt it using the icon next to the keys and entering your passphrase. (Again, make sure it is the PGP decryption.)

Comments:

The best known public key cryptographic system, RSA, was presented in the seventh lecture. It is one of the systems included in PGP. 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.

Exercise 7

Small numbers.

Start your Maple program. Type `restart;` at the beginning to make it easier to execute your worksheet after you have made changes.

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. 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.)

Exercise 8

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*, as described in the second lecture. 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 32, 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.

Exercise 9

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. 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, `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 10^6 and 10^7. With 10^6, it will probably take about two minutes, and with 10^7, probably about 5 minutes.

Exercise 10

Send your worksheet with Exercises 7, 8, and 9, including the result for 10^6 or 10^7 in Exercise 9 to your instructor. You should also have sent e-mail to your instructor in Exercises 3 and 5. Remember to logoff.