Project Description

Contents

1 Deadline, Logistics, and Interview

1.1 Deadline

The deadline for the project submission is 2015-04-28 11:59 pm EDT.

1.2 Questions

With any questions, please contact Sonali Malik, `mailto:sm5119@nyu.edu`, but if after communicating with Sonali Malik there are unresolved issues, please contact Zvi Kedem `mailto:zk1@nyu.edu`.

1.3 Submission Process

Information will be provided later.

1.4 Interview

You will be individually interviewed by Sonali Malik and by Zvi Kedem. You are, of course expected to understand completely your program and understand what the various algorithms do and how. The available dates and times of interviews will be set up later. As long as you submit the project by the deadline, if you have a good reason why you cannot make any of the available time slots, individual arrangements can be made.

2 Introduction

The project is somewhat involved, so we tried to provide a very detailed description. If anything is seriously unclear we will modify the project description. You must read the assignment carefully, so we recommend printing it out and checking that what you have actually done is what had been specified: do not just rely on reading the assignment on the screen.

Please follow the instructions precisely as given. If anything is unclear, please email Zvi Kedem. The project is somewhat underconstrained in order to enable you to make decisions as you find them convenient.

Please use the versions of the algorithms that we covered in class. For example, our version of Miller-Rabin, and not another one. However, if you prefer to use a different version of the Extended Euclidean Algorithm, you may do so. Please read the algorithms carefully. Relying on your memory may not be good enough.

Your program should be well documented, so it can be understood just by reading the source code. Similarly, the output of your program should be completely understandable just by reading it.

3 Overview of the Project

Your task is to build an RSA public/private cryptosystem and use it for creating digital certificates and authentication.

So that you get a good feeling of what is going on, you will not be relying on any libraries for this. You may use any programming language you like.

You are permitted to use a random number routine of your choice, i.e., you do not need to write your own (pseudo-)random number generator or create true random numbers. Make sure that if necessary, you set the right parameters for the routine, as in some cases if you do not do that, all the random numbers in one run of your program could be identical. The random numbers generated should be integers, so if you need a bit you need to extract it. You will also get a specification for a one-way hash function. It will be a very bad one-way function but sufficient for our project.

When you are done, what you will create is pretty close to what really happens, other than that, in order not to worry about precise arithmetic with thousands of bits, you will work with very small integers and also the challenge to be decrypted will not involve nonces (as they are used, e.g., in TLS). As you know, computations are done, in most cases, modulo some number n. In our setting, n will be so small that n squared will not overflow 32 bits. So if your (integer) variables are 32 bit integers and you take modulo n as soon as you can, everything should work without worrying about overflows.

So the difference between what you will do and what really happens is the need to create routines for precise arithmetic for very long integers (and to have good random numbers and a good hash function). It is easy but tedious to do on your own and it will take us too far from the goals of the course, and such precise arithmetic packages exist, but it is better if you do not rely on them so that you have a firmer grasp of the key issues.

You will probably find it useful to write some routines for getting random bits, multiplications and raising to powers modulo n, as these operations will appear repeatedly, etc. If your programming language has operations modulo n, please do not use them but write you own. As you will see, you will also find it useful to write some routines for manipulating bits. You can use whatever routines exist in your programming language for manipulating bits to help you create the routines you want.

The project is not very large but if you organize the programs/routines well, it will save you a lot of time.

Your assignment will be defined in paragraphs that are indented, just like what's coming next.

> Write a program to produce what is specified below. Your program should be well designed and not “just work”. Please comment what you are doing extensively, as you will “walk through the code” with Sonali Malik and Zvi Kedem.
>
> You will be also printing traces, please label the variables to be printed. So if you are asked to list k, please produce something like

`k = 11`.

You will also be asked to print the line number shown on the left in the instructions, so it is easy to find various parts of your submission.

4 Building an RSA system

To build an RSA crypto system, you first need to find two random primes. Your primes will consist of 7 bits. You will set the first and the last bits to be 1 and randomly choose one-by-one the internal 5 bits. We could have used only 7 bits to store it, but to simplify the program, store it as a standard 32 bit integer, so there will be 25 leading 0 bits.

You may use any pseudorandom routine you like to generate random numbers. Very likely, your programming language has one. To generate a random bit, get a (pseudo-)random number, and extract the least significant bit to be your random bit.

Print a line with the line number shown on the left and below that line, for one of the random numbers you got, produce a trace showing how the 5 individual bits were obtained, so show your 5 random numbers and the extracted bit for each.

You will use Miller-Rabin algorithm as taught in class notes (and not as it is generally presented) to test if your 7-bit number n is a prime. You will pick some random a's, such that $0 < a < n$. You can do it either by a process similar to getting a candidate prime above, or to simplify your program, you can just get a pseudorandom number and "cut it down to size" by computing its remainder modulo n, and discarding it if it is 0. If your number passes the Miller-Rabin test for 20 values of a, you may declare it as prime. For completeness, make sure that one of the n's turned out not to be a prime number. So, if you immediately find the prime numbers you need, pick any number you know not to be prime and perform the test on it. If the test says it is possibly a prime, look for another number that you know not to be a prime.

Print a line with the line number shown on the left and below that line, for one n that turned out not to be prime and the a for which the answer was "not prime," produce a trace, similar to what we had in class notes.

Print a line with the line number shown on the left and below that line, for one n that turned out to be prime and one a for which the answer was "perhaps prime," produce a trace, similar to what we had in class notes.

Now, given two primes p and q you found, you will get $n = p \times q$. Note that p and q should be different, so you need to check for this. In a "real" search for large number the probability that $p = q$ is so small that there is no need to check for this, but as you are working with very small numbers, you need to check for that.

Pick a small number to be the public key e. It has to be relatively prime with $\phi(n)$. To check if it is relatively prime and to find a multiplicative inverse to serve as the private key d, use the Extended Euclidean Algorithm, which you will code (or just take it from your previous assignment). If e is not relatively prime with $\phi(n)$, then find another small number. Do not do it randomly, start with 3 and go up until you find an appropriate e. In the extremely unlikely case (which would not happen in a real RSA implementation), that all the values of e you try do not work, just pick another random prime and start again. And if the smallest e you find is big, which here means bigger than $\sqrt{\phi(n)}$ that is also fine for your project.

> Print a line with the line number shown on the left and below that line, for the system that Alice (see later) will use, show how you found e, that is show how you used the algorithm on the various candidates for e until you got the right one. (If you are lucky, the first e you tried, worked—this is fine too.). Produce a trace for each e just as we had in class for the Extended Euclidean algorithm. For the value of e found, find the corresponding value of d and normalize it so that it is positive and smaller than $\phi(n)$). (If it happens that $d = e$, this will be fine too for your project, though of course not in a real RSA system.)

> Print a line with the line number shown on the left and below that line, list the value of d.

So, the public key of Alice, is really the pair $\langle n, e\rangle$ and the private key of Alice is $\langle n, d\rangle$; only she knows the latter—more precisely only she knows d.

> Print a line with the line number shown on the left and below that line, list the following for Alice, both as integers and as sequences of bits: p, q, n, e, d.

5 Creating a digital certificate for Alice

You will be three people: Trent, Alice, and Bob. We will not build parts that just repeat other parts, but will have at least "one of each" that's interesting.

You will create RSA systems for Trent and Alice. (You do not need to check that Trent and Alice will have different n's, as in a "real" RSA systems this guaranteed probabilistically.) Trent's public key will be known to all (Trent, Alice, and Bob). Trent will issue a digital certificate to Alice. The digital certificate will consist of two parts:

1. the pair $r = \langle \text{Alice, public-key-of-Alice} \rangle$
2. the signature $s = \text{Trent's-signature-on-}r$

Let us discuss the the formats and what needs to be done. The format for r will be as follows. Its length will be 14 bytes, defined as:

1. Bytes 1–6: The string Alice padded to the left with blanks. We assume that a name will fit in 6 bytes, but Alice only needs 5.

2. Bytes 7–10: n, padded with leading 0 bits, as necessary, so it is a standard integer stored in 32 bits. (Note, that n could have been stored in 2 bytes, thus mimicking closer what happens in reality when n is big, but let's not worry about this.)

3. Bytes 11–14: e, padded with leading 0 bits, as necessary, so it is a standard integer stored in 32 bits.

The signature s is $h(r)$ decrypted with Trent's private key (using fast exponentiation).

Let's talk about h, the one-way hash function. It will be 1 byte long. You will partition r into bytes and compute their exclusive or ($\oplus$). The resulting 1 byte will be your hash. It is simpler to store it as integer, so pad it with leading 0 bits. Therefore, s will be stored using 32 bits.

> Print a line with the line number shown on the left and below that line, list the following as sequences of bits: $r, h(r), s$.
>
> Print a line with the line number shown on the left and below that line, list the following as integers: $h(r), s$.

Alice is given the certificate. We will skip the step that Alice takes to confirm that she got a valid signature as signature checking needs to be done by Bob anyway, so we do not need to practice doing this here.

6 Alice authenticates herself to Bob

Bob is going to have Alice authenticated. He gets her certificate. We skip the checking of whether this certificate was signed by Trent, and checking that the format is right, etc.

Bob will pick a random number u, big, but smaller than n. He will do it as follows. Consider the bits of n, $n = n_{31}n_{30}\dots n_1n_0$. There are going to be some leading 0 bits as n really fits in 2 bytes. So let k be such that $n_{31} = \cdots n_{k+1} = 0$, but $n_k = 1$.

Let $u = u_{31}u_{30}\dots u_1u_0$. For k defined as above, set $u_{31} = \cdots = u_k = 0$ and $u_{k-1} = 1$. Choose $u_{k-2}, \dots, u_0$ randomly, bit by bit.

Example: $n = 10057$, which in bits is 00000000000000000010011101001001. From the value of n we see that $k = 13$. Therefore, u will be of the form 0000000000000000001xxxxxxxxxxxxx where all the x's are random bits, and $4096 \leq u \leq 8191$.

Print a line with the line number shown on the left and below that line, list the following as integers: k, u

Print a line with the line number shown on the left and below that line, list the following as a sequence of bits: u

Bob will send this number u to Alice. She will compute $h(u)$ and using fast exponentiation decrypt it with her private key getting v. She will send v to Bob. Bob will encrypt v with Alice's public key using fast exponentiation and check whether it is indeed $h(u)$. If it is, Bob knows that he is talking to somebody who knows Alice's private key.

Print a line with the line number shown on the left and below that line, list the following as integers and as sequences of bits: u, $h(u)$, $v = D^{\mathrm{RSA}}\bigl(d, h(u)\bigr)$, $E^{\mathrm{RSA}}(e, v)$. (e and d are, as computed before, Alice's.)

Print a line with the line number shown on the left and below that line, show a trace of your computation of $E(e, v)$ (using fast exponentiation, of course).