CS337: Project 2

TA in charge: Nam-phuong Nguyen (namphuon@cs.utexas.edu)

Release Date: Sept. 21, 2010
Due Date: Oct. 3rd, 2010, 11:59 PM

1 Introduction

Encryption is used when secrecy is desired, for example, in communication between two parties. The RSA encryption algorithm is arguably the most popular encryption algorithm in current use. It was invented in 1978 by Ron Rivest, Adi Shamir and Leonard Adleman. For this project, you will implement the RSA algorithm to encrypt and decrypt files. You are expected to be familiar with the algorithm as described in the class notes (Section 3.3.2).

2 Description

This project consists of three parts :

- Writing a program to generating a RSA public key and private key pair.
- Writing a program to encrypt files.
- Writing a program to decrypt RSA-encrypted files.

Each of these parts will now be described in detail.

3 Generating the RSA key pair

In this part of the project, you are required to create a RSA public key - private key pair generator. The inputs to your program are two primes p and q, $p \neq q$. The outputs are a triple of numbers n, e and d(in the same notation as the class notes). You will use your program to generate keys to test your encryption algorithm. When you test your program, you will need to make sure your choices of p and q are large enough such that n is between 2^{24} and 2^{30}. The purpose of this restriction is to allow you to do all the required computations for encryption and decryption using just the Java built-in *long* data type.

When your program is invoked with the arguments "key <p> <q>", it will output n, e, and d. An example input/output would look exactly like this:

$java -ea RSA key 47 59
2773 17 157

4 File Encryption

In this part of the project, you will implement the RSA algorithm to encrypt an arbitrary file.

How do you go about doing this? Note that the RSA algorithm specifies how to encrypt a single number ($< n$). To encrypt a file, it is sufficient to break up the file into blocks so that each block can be treated as a number and encrypted by the RSA algorithm. For this project, you will use a block size of 3 bytes. To treat each block as a number, simply concatenate the bit representation of the 3 bytes to form a single number. Note that this number will be between 0 and 2^{24} and so, will be less than n (why?), allowing us to to use the RSA encryption algorithm for a single number.

After encrypting the block, the encrypted number has to be written out to the output file. This number is between 0 and n, and therefore, potentially between 0 and 2^{30} . To write out this number, simply break the number into 4 bytes based on the bit representation of the number and write out the 4 bytes in order from *left to right*.

Let us now walk through the entire process with an example. Suppose the input file consists of the following bytes (in order) : 75, 34, 107, 23 . . . . To generate the first encrypted number, pick up the first block consisting of three bytes : 75, 34, 107. Form the number from this block by concatenating the bit representations of these bytes - 01001011, 00100010, 01101011. This number is 00000000 01001011 00100010 01101011. Now, encrypt this number by the RSA algorithm to get the number (say) 01011000 00110110 00001000 10001100. To write out this encrypted number to the file, break it up into 4 bytes, reading the bits from left to right - 01011000, 00110110, 00001000 and 10001100. Write out these 4 bytes in Big-endian (left to right) order to the output file.

5 File decryption

In this part of the project, you will implement the RSA algorithm to decrypt an encrypted file.

How do you go about decrypting the encrypted file? This procedure is the exact reverse of the encryption procedure. You will read from the encrypted file in blocks of 4 bytes, with each set of 4 bytes forming an encrypted number. The RSA decryption algorithm specifies how to decrypt an encrypted number. This will give you the number that was formed from the original plaintext file. Now, remember that this number was formed by concatenating 3 bytes from the plain text file. To get back those 3 bytes from this number, you will have to pick out the lower order 3 bytes from this number and write them out in order

to the output file (the topmost byte of this number will always be 0, can you see why?).

Let us walk through this procedure with an example. Suppose the encrypted file consists of the following bytes : 88, 54, 8, 140, . . . (these were the same bytes that were written out in the example for the encryption procedure). The encrypted number is formed by concatenating the bit representations of these numbers together - 01011000 00110110 00001000 10001100. Decrypting this number by the RSA algorithm gives the original number - 00000000 01001011 00100010 01101011. This number is then split into the original 3 bytes by taking the lower order 3 bytes in the same order - 01001011, 00100010, and 01101011. Write out these 3 bytes in Big-endian (left to right) order to the output file.

6 Submission

The key generation, encryption, and decryption will be handled by one class, RSA.java (that is, this file should have the main method). For encryption, the program will be called with the keyword encrypt, the input plaintext file, and the key file as arguments. The key file will be in the same format as described in Section 3.1, consisting of n, e, and d, in that order. For example:

Key file contents:
2773 17 157

The program will be invoked as :
$java RSA encrypt plaintext keyfile
where **plaintext** is the name of the file to be encrypted and **keyfile** is the name of the key file to use.

The encrypted file that the program generates **should** be named "encrypted".

For decryption, the program will be called with the keyword decrypt, the input encrypted file and the key file as arguments. The key file will be in the same format as described in Section 3.1.

The program will be invoked as :
$java RSA decrypt encrypted key.txt
where **encrypted** is the name of the file to be decrypted and **key** is the name of the key file to use.

The decrypted file that the program generates should be named "decrypted".

7 Points to take note of

Your submission for this project is complete if your submission for the first part creates a valid RSA key pair and if your encryption and decryption algorithms

work correctly as specified. Note that your program should stick to the specifications laid out in this description in all respects, from the order in which bytes are written to the order of the command line arguments of the program. **Your encryption routines should work correctly when used with other correct decryption routines and vice versa.** It is therefore necessary for you to follow every specification exactly. An easy way to test your program is to give your encrypted file and key to another group and see if they can decrypt your file and vice versa.

The RSA encryption and decryption algorithms involve modular exponentiation. Straightforward exponentiation will require you to manage really huge numbers that will be outside the limits of the basic Java data type 'long'. For this reason, it is necessary that **you use the algorithm for fast modular exponentiation** that is described in the class notes (Section 3.3.2). You **should not use any other large number data types like the Java BigInteger class or any similar built-in crypto libraries**. Any submissions that use such data types or libraries will face a significant penalty. The entire project has been designed so that all operations can be performed using Java's standard 'long' data type.

The DataInputStream and DataOutputStream have methods for reading and writing Java primitive data types. Using this will help allow you to not worry about the byte ordering, as it writes/reads the primitive data types in Big-endian order.

In addition, you can assume that the file will not end in a zero byte. For example, if there are only 2 bytes left to encrypt, with values 2 and 3, you can encrypt it as 00000000 00000010 00000011 00000000. This also applies when decrypting. When you decrypt to receive 00000000 00000010 00000011 00000000, you can assume that there are only 2 bytes left in the file, with values 2 and 3.

8 Questions and Clarifications

Clarifications regarding this project will be posted to Blackboard. Although changes are not expected, you are responsible for any modifications posted to the forums, so be sure to check it regularly.

9 Deadline and Submission Instructions

All the requirements in this project description document must be strictly followed to avoid points deduction.

The project is due Sunday, Oct. 3rd, 2010 11:59PM. Try to submit your work early to avoid any issues with the server, such as congestion, system-failure, etc.

9.1 Files to be submitted

List of the files you must submit:
(The names of the files should be exactly the same as follows and case-sensitive):

- emails.txt
  "emails.txt" content and format: (You MUST follow the exact format below! Otherwise, no comment and grade will be sent to your and your partner's emails):
  your CS account your email
  partner CS account partner email

  Example "emails.txt" (Note: There is a space between your CS account and your email):

  namphuon namphuon@cs.utexas.edu
  afdreher afdreher@cs.utexas.edu

- readme.txt (Do not forget Section#!)
  NAME of partners;
  Section # of partners;
  EID of partners;
  CODING STATUS: (very important!);
  IMPORTANT: Summarize your code status in readme.txt file. If your code works perfectly, explicitly saying this in the readme.txt. If you code does not work (cannot compile, cannot run, doesn't correctly decrypt, etc.), you need to specifically explain this: what have been done so far and what have not been done, what is working and what is not working to avoid confusion! Put proper comments in all the code files.

- RSA.java

9.2 Grading

I will be using the following commands to grade your project:
$javac *.java
$java -ea RSA key <p> <q>
$java -ea RSA encrypt <myfile.txt> <mykey.txt>
$java -ea RSA decrypt <myfile.txt> <mykey.txt>

Make sure that typing in those commands work. You will be heavily penalized if I cannot compile your code.

9.3 Turnin

All these files should be turned in to 'project2', to the grader 'namphuon'. A sample turnin command line is

$turnin –submit namphuon project2 <file-name>

To submit your project, use Linux "turn-in" program exactly as follows.

$turnin –submit namphuon project2 RSA.java readme.txt emails.txt
To confirm your submission:
turnin –list namphuon project2
ONLY ONE submission per group!

- Please do not turn in a tarred or compressed version of your files.
- Please strictly follow the files names. Otherwise, your submission may not be graded.
- You must make sure your program will work correctly on CS Linux machines.
- All of your text files, such as the readme.txt, etc., should be generated using CS Linux. If you don't use a *nix system to generate these files, make sure you perform the correct encoding conversion prior to submission.

No changes are expected to the project description, but be sure to watch the newsgroup just in case there are any. You are responsible for noting any such updates. All the best working on this project!!