CS1713 ALGORITHM DESIGN AND ANALYSIS

# Programming Assignment 2

## Idea

To use a simple RSA envelope with a Vigenere cipher to encrypt and decrypt files.

## Method

You will write one program with three main parts:

**makeKeys** – using the algorithm discussed in class this will generate values p, s, N for an RSA encryption system. Print out p, s, N after they have been created and then store p, N in a file called **pubkey.txt** and s, N in a file called **privkey.txt**. You must make sure s is between 0 and PHI by adding on or subtracting PHI.

**encrypt** – this will generate a random Vigenere cipher consisting of 32 byte values (each between 0 and 127). This byte array will be converted into a large integer using Horner's Method (x = 128) and encoded using RSA (the public RSA key will be read from **pubkey.txt**). The encoded RSA integer will be written to the file **Big.txt**. Then you will encrypt a source (text) file using the Vigenere cipher. In the encryption you will successively add on (mod 128) one element of the byte array to a character. So the first character will increase by **vigKey[0]**, the second character by **vigKey[1]** etc. You will need to cycle around to the beginning of the key – for the 33rd character add on **vigKey[0]**, 34th character add on **vigKey[1]** and so on. As you encrypt, the bytes will be written to an output file.

**decrypt** – this will read the RSA encoded Vigenere Key from **Big.txt** and decrypt it (using the private RSA key read from **privkey.txt**). Then, by reversing Horner's Rule, you will recover the Vigenere Key (as a byte array). When decrypting the message, you should subtract (mod 128) successive elements of the byte array from the bytes in the output file from **encrypt** and you will recover the original text of the source file.

#### General Notes:

1. You can hardcode in the filenames **Big.txt, privkey.txt, pubkey.txt.** Other filenames should be input by user.

2. The initial random primes generated (x, y, and p) should be of length 500 bits.

3. Horner's Rule: we discussed in class how to implement this by an iterative process of addition and multiplication using x = 128. Remember that **VigKey[0]** would be the constant term and **VigKey[31]** the coefficient of 12831.

To reverse Horner's rule:

for i = 0 to 31

VigKey[i] = VigKeyInt % 128

VigKeyInt = (VigKeyInt – VigKey[i] )/128

**Java Help**

1. Use class **BigInteger**: As its name suggests this is used to store very large integers such as those used in RSA. In this project you will store the RSA keys, the converted Vigenere Key and its RSA equivalent as **BigInteger** objects. You will also need to use a variety of methods from the **BigInteger** class to instantiate objects, do arithmetic and modular exponentiation. You can get all the information from the Java API. In particular use the static method **probablePrime** to generate primes for x, y, p and the method **modInverse** to generate s from p and the method **modPow** when encrypting and decrypting. Remember that Java does not allow operator overloading so +, -, \*, / will not work with **BigIntegers.** Instead there are methods for carrying out all these operations and constants for BigInteger versions of 0 and 1.

2. Writing to and Reading from Files: When dealing with **BigInteger** objects use chaining of streams:

**// f is the file to be written to**

**ObjectOutputStream outObject = new ObjectOutputStream(**

**new FileOutputStream(f));**

**// Use method writeObject to write objects to file f**

Reading objects is done in a similar way except that the object must be cast to a **BigInteger** when it is read from the file.

To read primitives from a text file/command line, use the Scanner class and to write primitives to a text file use the Formatter class.

3. Exceptions: Since we are dealing with writing to and reading from files you must handle exceptions **ClassNotFoundException** and **IOException.**

4. **Data Types and Casting**: **byte** and **int** are primitive data types in Java and you will often need to switch between them in this assignment. Java will automatically promote a **byte** to an **int** but you must cast an **int** to a **byte**. You will also need to use methods in **BigInteger** to do some switching between data types, for example when using Horner's rule.

#### Turning in the Assignment

#### This assignment is due on Monday 14 October.

1. Print out your Assignment Information Sheet and all source code. Put them in a folder and turn in.
2. Upload all source and compiled files to CourseWeb (preferably zipped). **Please include a test file called test.txt** of about 200 – 300 words.