CSC 207 **Assignment 6: Implementing RSA** **Fall 2009**

Due: Friday 13 March 2009

Topics: I/O, Recursion.

Collaboration: This homework assignment may be completed individually or in pairs.

Submission: Follow the instructions for submitting programs via P-Web and handing in a printed copy. Be sure to generate test cases and a statement justifying their sufficiency. One only one submission (paper and digital) per group is necessary.

We have studied recursion and some of the mathematical theory behind the RSA encryption algorithm. Your task in this homework is to create a working implementation of the RSA algorithm to encrypt and encrypt messages. You will write a set of three programs

- A key generator for generating public and private keys
- An encryption program for creating an encrypted file using the public keys
- A decryption program for decrypting an encrypted file using the private (and public) keys

Your suite should include any appropriate classes and exhibit good encapsulation and cohesion, making use of design patterns as appropriate.

Key Generator

The usage for your key generator should be

```
java RSAKeygen bitLen keyname
```

The output of this program should be two files, `keyname` and `keyname.pub`, where `keyname` contains the private key (everything needed to decode a message) and `keyname.pub` contains the public key (*only* everything needed to encode a message, nothing more).

It may be helpful if you use Java's serialization to store (and later load) objects as files. To do this, create an object class whose only members are objects that are serializable (i.e., `implements Serializable`). Java can then take care of saving your object (i.e., its members) to disk for you (see the Java API for `Serializable` for details on how to save and load an object using `writeObject` and `readObject`).

Your key generator should use the algorithm described in Weiss section 7.4.4 to create the appropriate values. All integers and mathematical operations on them should use `BigInteger` objects. Note that the `BigInteger` class provides all the methods you will need.

Encryption Program

The usage for your encryption program should be

```
java RSAEncrypt keyname.pub < message.txt > encrypted.dat
```

In other words, the only command-line argument should be the name of the file containing your public key information. The message will be read from standard input (`System.in`) and the encrypted version written to standard output (`System.out`). You will need to convert the input message to raw bytes and interpret this as a number. To do this, note that `System.in` is an `InputStream` object, which will allow you to read individual bytes from the stream using the `read()` method. The `BigInteger` class has a constructor that takes a `byte[]` array.

When you have encrypted the message (a `BigInteger`), you must write it to the encrypted file. You may do this one of two ways:

- Write the encrypted result explicitly as a sequence of bytes (See `ByteArrayOutputStream` and its `writeTo`) method.
- Use Java's serialization to write the encrypted result.

Whichever you choose, your *de*cryption program must be able to read the result as a file from standard input.

Decryption Program

The usage for your decryption program should similarly be

```
java RSADecrypt keyname < encrypted.dat > decrypted.txt
```

Once again, the only command line argument should be the name of the file containing your private key representation. Note that this will have to store everything you need to decrypt the encrypted message. The encrypted message will be read from standard input (`System.in`) and the decrypted version written to standard output (`System.out`). Note that, depending on which method you used to write the encrypted file, you may need to:

- Read the encrypted result as a sequence of bytes (*you've already implemented this!*)
- Read the result as an object (see the `ObjectInputStream(InputStream)` constructor)

Your decrypted message (using the algorithms in the book) will again be a `BigInteger` object. You will need to write this as a series of bytes to `System.out`, which should end up being exactly the same as the original file `message.txt`.

Here is a good way to test your program:

```
java RSAEncrypt keyname.pub < message.txt | java RSADecrypt keyname > message2.txt
diff message.txt message2.txt
```

Of course, since you're reading and writing messages as bytes, you can encrypt *anything*, word processor files, images, etc. Try it!