Author: Justin Luttrell

RSA Cryptosystem

Overview:

This project is an implementation of the RSA Cryptosystem. It contains three separate modules Key Setup, Encrpytion, and Decryption. The project is written in java and makes use of the BigInteger class to perform calculations of large integers. For compilation and execution instructions, see the end of this document.

Components:

Key Setup Module:

Associated Files: KeySetup.java, KeySetupLauncher.java, & ModularExponentiation.java

The key setup module is responsible for generating the public and private keys that will be used for encryption and decryption of messages.

The KeySetup.java file is the implementation of the KeySetup class that extends the ModularExponentiation class. The methods in the KeySetup class include:

1. generatePrime() to generate a large prime number
2. fermatPrimalityTest() to test whether or not a generated integer is probably prime
3. extendedEuclidsAlgorithm() to compute the multiplicative inverse, more specifically used to compute d (private key).

The KeySetupLauncher.java file contains the main method for creating public and private keys. It generates the keys as outlined by the RSA algorithm and makes use of the methods in the KeySetup class to do so. There is no input to this module, however this module outputs public_key.txt and private_key.txt to the main folder of the project directory. The public_key.txt contains the n value followed by a single space then the e value. The private_key.txt contains only the d value.

Encryption Module:

Associated Files: Encryption.java & ModularExponentiation.java

The encryption module is responsible for encrypting a numeric (space-free) message that the user has. The Encryption.java file contains the main method for this module as it takes as input the public_key.txt (generated from the key setup module) and message.txt. It gives as output the ciphertext in a file called ciphertext.txt. The encryption itself is performed by calling the modularExponentiation method.

Decryption Module:

Associated Files: Decryption.java & ModularExponentiation.java

The decryption module is responsible for decrypting the ciphertext (ciphertext.txt) generated from the encryption module. The Decryption.java file contains the main method for this module as it takes as input the public_key.txt, private_key.txt (generated from the key setup module) and outputs the decrypted message in a file called decrypted_message.txt. The decryption itself is performed by calling the modularExponentiation method.

Modular Exponentiation:

Associated Files: ModularExponentiation.java

The ModularExponentiation.java file contains the implementation of the ModualarExponentiation class whose sole method is modularExponentiation(). All three of the modules make use of this method when needed.

Testing

The test run and its associated files can be found under the test_run folder under the main project folder.

For testing of this project I made use of some online resources. The online resources allowed me to verify correctness of the algorithms I implemented. The method tested and the resource I used are as follows:

- modularExponentiation():
  - https://www.dcode.fr/modular-exponentiation Used this modular exponentiation calculator to verify that my modular exponentiation algorithm was implemented correctly
- fermatPrimalityTest():
  - https://www.alpertron.com.ar/ECM.HTM Used the integer factorization calculator to determine if my fermatPrimalityTest() was returning possibly prime correctly for given values.
- extendedEuclidsAlgorithm():
  - https://www.dcode.fr/modular-inverse Used this multiplicative inverse calculator to verify that my private key was being calculated correctly.

Problems Encountered

The biggest issue I encountered was in implementing the modular exponentiation method. This was the first algorithm I coded, so there was a learning curve involved with doing computations

while using the BigInteger class and I kept getting errors associated with improper use of its methods.

Another problem I encountered was in reading the text files and storing the results. I originally wrote to text files by using println(), however when parsing the contents of the files I realized the println was appending the '\n' character to the text file and so I decided to switch to the print() function to write to files to avoid the error of invalid declaration of a BigInteger.

LIMITATION: As described in the paragraph above, the text file should not contain any special characters, escape characters, etc. I have noticed that copy and pasting numbers into the text files to be used in the encryption or decryption modules can result in these special characters being included into the text file and this may produce erroneous results. Be careful about the way the contents of the files are input especially message.txt since this is the only file that the user creates themselves. Other than this as long as the message.txt is under the main folder of the project folder, all modules should work.

Compilation & Execution

For this project I created bash scripts for each module to make compilation and execution as easy as possible, therefore the project should be used on a Linux Terminal. When downloading the .zip file you may have to make each script executable. This can be done with the following command:

```
chmod +x [filename].sh
```

You can check the permissions for each file in the directory with the command:

```
ls -l
```

Execute Key Setup Module:

To execute the key setup module, from the main directory of the project folder run the command:

```
./KeySetup.sh
```

This will produce the two key files in the main directory of the project folder.

Execute Encryption Module:

To execute the encryption module, first ensure you have a valid message.txt file and public_key.txt file located in the main directory of the project folder and then run the command:

```
./Encryption.sh
```

This will produce the ciphertext.txt file in the main directory of the project folder.

Execute Decryption Module:

To execute the decryption module, first ensure you have a valid ciphertext.txt, public_key.txt, and private_key.txt files located in the main directory of the project folder and run the command:

```
./Decryption.sh
```

This will produce the decrypted_message.txt file in the main directory of the project folder.

A copy of this project can be found on github at www.github.com/jglu225/RSACryptosystem. You can direct any questions to me via email at jglu225@uky.edu.