CIS 181 Lab 01

Coding Review and Asymmetric/Public Key Encryption!

In this lab assignment, you will be dusting off your knowledge of programming from CIS 180 (or other equivalent course). Additionally, you will be utilizing a form of encryption known as asynchronous key encryption. Previously you may have been exposed to synchronous key encryption. With this type of encryption both parts that are passing information back and forth each have the same secret key that is used for both encrypting and decrypting data between end points. It's fairly simple, elegant, and relatively computationally inexpensive solution to encryption. One of the major issues though lies in *HOW* the two parties can obtain the key in a way that can't be intercepted?

A Simple, cumbersome solution, to symmetric key sharing

One solution would be to physically pass the encryption key on a USB drive to the other party. The drawback to this is the lack of flexibility. It is predicated that the two parties are geographically "close" to each other. It also makes changing the key periodically rather cumbersome. You could use the snail mail system, but that would also be a cumbersome solution, say nothing of the fact that it can easily be stolen (porch pirates!).

A Better solution to symmetric key sharing

A better solution is send the secret key over the internet, but encrypt it… wwwwwaait. Don't you need the key to encrypt it in the first place? Yup, so this seems infeasible except for situations you are just updating the key between two parties. There must be a way to encrypt something when you don't have a key (as described above) already shared between two parties. The answer, as I'm sure you've surmised at this point, is asymmetric key encryption or public key encryption.

How does public key encryption (asymmetric key encryption/cryptography) work?

With public key encryption, two keys are generated at the same time. 1. A ***public key*** for encrypting. 2. A ***private key*** for decrypting. The ***public key*** is given to anyone that requests it. They can use the ***public key*** to encrypt their data (a secret symmetric key) and send it to the party that generated the ***public key***.

Now despite having the ***public key***, knowing the process of encryption, and having the original unencrypted data, they cannot use the ***public key*** to decrypt the data. For instance, if another party intercepts your data you encrypted with the ***public key***, they cannot decrypt it with that same ***public key***, no more than you can. Only the original party, that still has the ***private key***, can decrypt the data.

Why not use public key encryption all the time then?

The simple answer: it's not particularly efficient when compared to symmetric key encryption. This is to say that it's a bit more computationally expensive to do. Therefore, it's preferred that public key encryption be used to establish a shared symmetric key between parties. Do the inefficient thing once, so you can do the more efficient thing from there are in.

What do you have to do?

You are given a number of files to use as outlined below:

- **PublicKeyEncryption.java**: This is a complete class that you cannot modify. It contains the basics to use Asymmetric Key Encryption.
- **SimpleAsymKeyEncryptApp.java**: This is *mostly* a complete class. You will need to add code and modify existing code as outlined below in steps 1-11.
- **MasterPrivateKey.key**: This is a private key that was generated along with it's public key equivalent. The public key was used to encrypted the next file…
- **Encrypted.bin**: This file contains encrypted text. The key in the file MasterPrivateKey.key will decrypt it.

Changed you need to implement:

1. Convert the ***for*** loop in ***private static String getTextFileAsString()*** method to a ***while*** loop. Make sure to comment the method as well.
2. Surround the long if/else structure in ***main*** method with a ***do*/*while*** loop, such that the program will only end when the user enters in the value *8* as a choice.
3. Convert the long ***if/else*** structure to a switch/case. Also take care of the scenario where the user does not enter a valid choice (1-8). Tell them they made an invalid choice. Hint: look up the default keyword.
4. For choices *1* through *7*… slightly modify the code to allow the user to choose the names of the files (instead of them being hardcoded). Hint, use an instance of Scanner to get the users response, and store their answer in a String.
5. Change choice *7* so the decrypted text is written to a text file (of the user's choice) instead of the console (Hint: replace ***System.out*** with an instance of ***PrintWriter***… don't forget to close it either!)
6. Test the program. Create a text file (either with notepad or Eclipse) and put a message in it (less than 117 characters). Run the program, choose *4* and encrypt the text file.
   Immediately choose 5. See if the decrypted text file matches the original text files' message.
7. Choose options *1* and *2* to save the current pair of public and private keys to text files.
8. Choose *8* to quit.
9. Rerun the program.
10. Use the private key provided to you **MasterPrivateKey.key** on myCourses with choice *7*. Use the file **Encrypted.bin** from myCourses as the encrypted file to decrypt here. The resulting file, if all works properly, will contain a message in English.
11. Rerun the program again a few times, choosing option 4 to encrypt a new text file with varying amounts of text. Be sure to save the encrypted files with different names.
12. Answer the following questions in the comments at the top of your code (**SimpleAsymKeyEncryptApp.java**)
    a. For each of the encrypted files you created, what can you note about the file size? (right click on the file and choose properties).

b. Does it seem like the amount of text you encrypted seemed to have an impact on the encrypted file's size?
c. Does the size seem to relate to the limitation of the number of characters in the textfile (117)?
d. If we can't go above 117 characters to encrypt, but we have more than 117 to encrypt, what would be a simple solution to fix this? (Hint: If you have to ship a large item, but that item is bigger than the biggest single box allowed to ship it, what can you do to the item to ship it and make sure it all makes it there in the end?)

13. Done? Demonstrate the program to the TA. Be sure to show the decrypted message from step 10 as well.
14. Submit the following files to myCourses.
    a. PublicKeyEncryption.java
    b. SimpleAsymKeyEncryption.java
    c. The text file you created with some message in step 6.
    d. The encrypted file created by the program in step 6.
    e. The private key file from step 7.
    f. The public key file from step 7.