Trevor Parsons

Project 2

Task 1

I used the Java Security and Java Crypto libraries for this project, but I did have to download the Java Cryptography Extension Unlimited Strength policy files to create 256-bit AES keys.

Task 2

For the LCG random number generator, I used the generator built into the Java Random class. For the cryptographically secure random number generator, I used the built-in function of the SecureRandom class.

Even though my range of random numbers is 0 to 2,147,483,647 and I calculated the GCD of 10,000,000 random integers for each test, I was not able to get my approximations of pi much closer to 3.05xxx. Additionally, the Random class generator would sometimes outperform the SecureRandom generator. This is some sample output from the program:

Estimated value of Pi for LCG Random Generator: 3.050202795894099

Estimated value of Pi for Secure Random Generator: 3.049812516981508

Task 3

At the beginning of the program, I generated two files, one of size 16 bytes (128 bits) and one of 1,000,000 bytes (1MB) full of random bytes. I then created 128- and 256-bit keys using SecureRandom value for the KeyGenerator, and then used that to generate two SecretKeys. For the AES encryption and decryption cycles, the shorter file was much faster as expected, as shown by some sample cycles per second output:

Number of Short File 128-bit key AES cycles: 414

Number of Short File 256-bit key AES cycles: 428

Number of Long File 128-bit key AES cycles: 43

Number of Long File 256-bit key AES cycles: 44

It was interesting to me that the longer bit keys would outperform the shorter keys in terms of number of cycles, but as I discussed with you it is not too surprising. For the RSA encryption, I chose to encrypt the file with the same AES 256 bit key that I had used earlier, and then encrypt the key using RSA. I was having trouble getting the Long File properly formatted to encrypt entirely with RSA, so encrypting the AES key was a workaround. As expected, the additional RSA encryption of the key did reduce the number of cycles/second, producing the following sample output:

Number of Short File RSA cycles: 111

Number of Long File RSA cycles: 37

Seeing that the Long File encryptions were far more efficient in terms of number of bytes encrypted/decrypted per second, their results multiplied by 2 can be used to estimate how large of a file could be decrypted in a second (as they would not have to do any encrypting):

-For the 128-bit AES key, 43 cycles \* 2 \* 1MB = 86 MB/second decryption

-For the 256-bit AES key, 44 cycles \* 2 \* 1MB = 88 MB/second decryption

-For the 256-bit AES key with RSA key encryption, 37 cycles \* 2 \* 1MB = 74MB/second decryption

If it was necessary to brute force the any of the files using the 256-bit AES key, it would require testing 2^256 possible combinations, or more likely, on average 2^255 combinations. For the example, I’ll assume a computer that can do 2 billion operations per second is trying to brute force the key, in which case it would require 2.88e+67 seconds, or 9.18e+59 years to complete the task on average.