Project Phase 2 – Due Tuesday, February 28

This phase of the project starts with two exercises in which you explore symmetric cipher availability and performance in the JCA. The results of the exploration should be put in a simple experiment report with two sections, as described below. The report should be turned in (hardcopy) at the beginning of class on the due date.

All Java source code must also be submitted by sharing a Bitbucket-hosted git repository with me (use my regular UNCG email address to add me to your repository's access list). You will need to create a Bitbucket account for this, if you don't already have one — this can be done at `http://bitbucket.org`. NetBeans has good support for git version control, and we'll briefly review the basic operations in class. Code will be graded on code quality and design as well as whether it functions correctly, so don't write sloppy code!

1. Write a Java program named `ExploreJCA.java` that discovers what ciphers (service type "`Cipher`") are available on a system using the `java.security.Security` and `java.security.Provider` classes. Print out all available ciphers in a readable form, for each one giving both the provider and the service/algorithm.

   The system you explore could be as simple as a regular desktop system with Java, Standard Edition (JSE), which can be done using NetBeans in any UNCG lab or on your own system. If you are feeling ambitious, you can do this for Android and see what algorithms are available there — the JCA exploration and enumeration code is the same, but figuring out how to report the results will take some thought.

   The first section of your experiment report should start with information about the environment you used (both system description and version numbers) — the goal is to give enough information so that someone reading the report could replicate your experiments if they wanted. After describing the system, give the list of algorithms that your code reported.

2. From the list of supported ciphers, you should benchmark AES and one other symmetric cipher (these should be two different symmetric ciphers, not just two different modes for the same cipher). Write two methods in a file `JCATime.java`, one to time single-key encryption throughput and one to time key testing rate, and use the NetBeans profiler to time your code as follows:

   - Your method to time single-key encryption throughput should initialize the Cipher with a key, and then send as much data through the cipher as is required for it to

run for at last 10 seconds (the data doesn't matter, and you can just throw away the ciphertext!). Don't waste memory by trying to hold all input data in a single array. Instead, allocate a decent-sized array (say, 100,000 bytes) and repeatedly encrypt that. You'll need to experiment with parameters to determine how much data to encrypt, but once you've got that value set you can divide the number of bytes encrypted by the number of seconds that your code runs to find the encryption throughput.

- Your method to time the key testing rate should loop through different keys, and encrypt a single block with each key. The number of keys you test should be sufficient for your code to run for at least 10 seconds, which will give you a reasonably accurate value for the key testing rate measured in keys/second. Each test should use a different key, so you'll need a way to step through different keys.

Remember that you will be graded on good coding practices as well as your results! This means that your benchmarking code must be general purpose — do *not* write different code to test different algorithms! You should also use named constants for parameters like the amount of data to encrypt, etc. Don't hardcode numbers in code statements!

The second section of your experiment report should start with a description of the system you used for testing. Since you are reporting on performance, you should report hardware specifications in addition to software versions. These specifications should include, at the very least, the precise processor model and clock speed. The performance values should then be reported in a table, with rows for algorithms and columns for the two tests. Remember to specify units for any measurement!

3. At the heart of the secure chat application is a class (named `Conversation`, for example) that sends and receives encrypted messages — that is what you are to write for this part of the assignment. The precise definition of how this should work depends on the outcome of our phase 1 design discussion (planned for Thursday, February 9). Regardless of the exact requirements, make sure you give careful thought to good, modular design using multiple classes. For example, there should be a class that is responsible for managing keys for a conversation that is initialized when the conversation is established. We won't cover how to establish these keys until later in the course, so for now this class should associate a key of all zeros with every conversation. This class can be updated in a later phase, and if you're careful then the `Conversation` class will be usable with little or no changes.