Assignment 2: Due Thursday, March 6

1. For the first part of this assignment, you are to become familiar with the TPM/J software and our use of Subversion within the lab to keep a repository of software.

   Subversion is a popular "version control system" — a system that tracks changes to a set of files. For our work on TPM/J, we'll use Subversion to keep several main "branches" of TPM/J code that we're working on: A "distribution branch" which holds the code exactly as distributed from the original authors, a "SPAN working version" that contains changes that are made available to everyone in the lab, and your own personal working version. You can run Subversion on any system you'd like to get and manage your code, using either the text or a GUI interface, but unless you have a machine with a TPM it would be best (by far) to do your work on the SPAN Lab machines.

   Getting started: The first thing you need to do is "check out" a copy of the code repository with this command:

   ```
   svn co https://span.uncg.edu/svn/tpmjwork
   ```

   The first time you run this it will complain about the Certificate from the SPAN server, but if you tell it to accept this certificate permanently it will keep it as a "good certificate." You will need to enter your SPAN account login information next, and then it will create a directory "`tpmjwork`" that contains the current code repository. Inside this you'll find subdirectories "`dist`" (for the distribution branch of TPM/J), "`span`" (for the SPAN Lab version), and "`users/youruserid`" for your own copy.

   Making your initial copy: Before you change any code, you will need to copy it into your space. To do this, from inside the `tpmjwork` directory, issue the command:

   ```
   svn copy dist/tpmj-alpha0.3.0 users/youruserid
   ```

   Saving any changes you make: If you make changes to code and want to "check in" your modified version, you can go into a directory that contains all changes (for example, in `tpmjwork/users/youruserid/tpmj-alpha0.3.0`), and issue the command

   ```
   svn commit
   ```

   This will bring up an editor for you to type some comments that describe the changes you've made to this revision, and then the changes are copied up to the server. By default

this editor is "`vim`" on the SPAN Lab machines, but you can change the environment variable "`EDITOR`" to make it use whatever editor you want.

Adding and removing files and directories: If you add new files that you want included in the repository and tracked by subversion, you need to use the Subversion "`add`" command. For example, if you wanted to add the file `newfile.java` in the current directory, you'd use

```
svn add newfile.java
```

Similarly, to add a new directory, use the Subversion "`mkdir`" command, and to remove files or directories, use "`rm`".

Finally, what to do for this homework problem: First, do the steps above to set up the subversion repository for your work. Then create a new directory called "`scripts`" under `tpmj-alpha0.3.0` and create a file `README.txt` in this directory where you leave a brief message (anything for now). Make sure you set it up so Subversion tracks this directory and file, and then commit your changes to the repository.

2. TPMs perform a variety of operations, and most of the time in these operations comes from the time required for basic underlying cryptographic operations. We would like to be able to have an accurate "timing model" of a TPM so that we can evaluate the efficiency of algorithms without having to implement them, and so that we can estimate the time a TPM would need in order to perform operations that we could proposing adding to a TPM.

   You should first write up a plan that includes how you will create your model (what are you going to time and how are you going to time it) and, just as important, how you will *validate* your model. For example, you could select certain TPM operations based on the cryptographic operations they perform, and time them to deduce the times for the basic cryptographic operations. Then you could use these results to estimate the time for operations you have *not* timed, and see if this agrees. For possible operations to use, you can use the information in our book, or in the TPM specifications (see the class web site), or in the TPM/J code. For the planning part, you may discuss ideas with others in the class, but the final design and plan should be yours.

   Finally, following your plan, you should create and validate your model using the SPAN Lab machines. If your plan requires creating any scripts or additional code, be sure to check the code in to the code repository when you're finished. Your assignment submission should include a final description of your timing model along with the values you have measured.

3. Cryptography Notes: Problem 3.1.

4. Cryptography Notes: Problem 3.3 (the example we cover in class is close to a solution for this, but it doesn't satisfy the requirements perfectly; a small modification works, but try to come up with a solution that is more significantly different). The most important part of this problem is to provide a complete and rigorous proof regarding the security of your construction — not a "sketch" of a proof, or an "outline", but a full and complete proof.

5. (*Graduate students only*) Cryptography Notes: Problem 3.9.