In getting started, it was evident through the java API documentation more than adequately supported the cryptographic functions that the project would require. The first step, and the hardest portion, of the design were determining the bounds of the application in terms of inputs and outputs. The second step was verifying that the cryptographic functions behaved properly and expectedly. The last step was creating a comprehensive graphical user interface that was both simplistic and easy to use.

When we created the cryptographic support functions as described in the project description, it was a bit unclear as to whether or not the HMAC was a resultant of the ciphertext or the unencrypted text. We also recognized that given we only write the HMAC and the ciphertext to the file, it is impossible to determine a java decryption failure is a result of a mutated file or an incorrect password. Once we received word from the teacher that we indeed cannot determine between the two issues, then all fears were alleviated and the program could be called close to finished. When we first tested the cryptographic functions, there was one big problem in that the hmac-key appeared to have different lengths instead of being constant. This was a resultant of repeated transcoding of the bytes of text to a UTF-8 format that java did not use properly in the cryptographic hash functions. Removing the transcoding and leaving the bytes as is solved the issue right away.

As for the UI, we chose a highly simplistic approach which is composed of a singular textArea field on a JPanel and a small list of IO operations available to the user in the File Menu. To further facilitate user’s actions in the system, each file menu action has a system shortcut enabled and is easily viewed on the file menu bar. On the backend of the UI, we wrote efficient logic to validate user actions when performing different tasks. If there are ever any problems with a user’s efforts to change the system, then the user is immediately notified with verbose popup messages that explains the errors. The hardest part about the UI was finding a way to create a masked password input box. After a bit of clever googling, we found an amazingly easy solution to the problem that required only 4 extra lines of code in the program. The only problem with the solution is that the input password text field is not the default focus when the panel is shown, which forces the user to press tab each time he wants to begin typing his password, or manually select the box with his mouse. The result is less elegant than the ideal solution, but it works as it should.

We both initially started off working on the project on our own nearly as soon as it was assigned. Five days before the project was due, Joe asked Eric if he would like to team up since Joe was missing a few UI elements and Eric had several great looking UI elements, but needed some crypto support backends. Joe was supposed to deliver these libraries to Eric within a day or two to allow for ample time of integration, but he failed to follow through with that promise and was only able to deliver the libraries by late Sunday afternoon. As a result, it was decided that the simplest current working solution would be used in favor of saving time and not breaking anything in the project. Both members worked diligently on their parts, but due to Joe, integration fell through at the end and the UI remains simplistic yet 100% functional. This paper was a collaborative effort from both Joe and Eric.

# How to use the program:

Compile the following source files with javac:

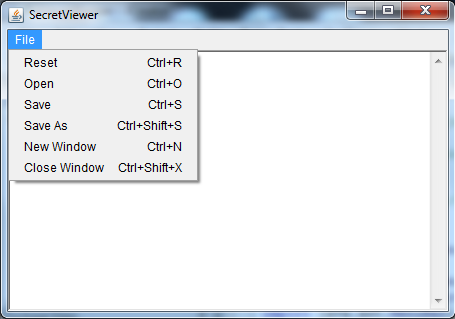
Salts.java

MyHash.java

MyCrypto.java

SecretViewer.java

Then run the program with “java SecretViewer” in the class path.



|  |  |
| --- | --- |
| **Command** | **Effect** |
| Reset | Closes any active document and clears the textArea field. |
| Open | Closes any active document and shows a file selection pane. |
| Save | Behaves exactly like “Save As” if there is no currently open document. Otherwise, it will prompt the user for the password of the existing file before attempting to commit any changes that the user has made. |
| Save As | Regardless of whether or not a file is open, this command will prompt the user for a location to save a new file and then prompt for a new password to encrypt the new file with. If anything fails during this step, then the state of the application is preserved. |
| New Window | Opens up another window instance of the program. |
| Close Window | Closes the current window instance of the program. |