Workflow Separation
We worked in pairs: implementation (Ruixin+Louise), unit+integration testing (Brandon+Zilong). We also performed pair programming, especially when it came to finding some of our more subtle bugs.

Testing of Source Code
We performed unit testing for Validator and Encryption. The Validator unit tests are the same as those we had for Milestone 3, with additional test cases for edge cases.
For this milestone, we also:

- Performed manual testing to verify DB entries correspond to functions called
- Preparing shell script/bat file for integration testing with DB
- Worked on different OSes (Windows 8.1/Ubuntu 16.04) to ensure project works as intended for both OSes (Differing OS level security restrictions troubleshooting)

We performed testing for all encryption functions which we used in our code to ensure confidentiality and integrity. Testing can be found in the test_Encryption.java file.

- Some of the test inputs were generated using random strings with seeds for reproducibility and to cover a wider range of tests
- Performed unit testing on independent encryption methods such as salt generation and password hashing
- Performed integrated testing on dependent encryption methods such as salt generation, IV generation and asymmetric/symmetric encryption
- All encryption and decryption methods were tested to ensure that the same object will be re-obtained after decryption

We also performed manual testing for the following components:

- Setup of server
- Downloading file logs as admin
- Downloading user logs as admin
- Deleting a user as an admin
- Login
- Registration
- Upload of file
- Creation of folder
- Sharing of file/folder
- Renaming of file/folder
- Deletion of file/folder
- Overwriting of file/folder
- Viewing file logs as viewer/editor
- Changing password of a user
- Changing email of a user
- Deleting a user account as that user

We checked that the actions were correctly prohibited if the user did not have the rights to perform that action. We also checked that sharing a folder also shared the contents of that folder correctly. We also tested this on different computers with several clients logging in to the same server. We also tested our executable jars on the command line.

FindBugs
A FindBugs analysis on the almost final product (minus a few bugfixes) revealed 224 bugs. 5 bugs were in the scary category (ranks 1-9), another 2 bugs were in the troubling category (ranks 1-14), and the rest of the bugs were lower ranked.

4 of the rank 5 bugs were caused by ignoring the result of java.sql.PreparedStatement.executeQuery(). This was due to the fact that the function did not return a value regardless of success or failure. This was fixed by returning a boolean true in the event of success, and further making a check after the call of the functions to respond to the success or failure accordingly.

1 of the rank 5 bugs was a null pointer dereference of createLog in org.cs5431.SQL_Accounts.changePassword(JSONObject, String, String). This was a genuine correctness error that was fixed by moving the initialisation of createLog outside the if/else statements. The rank 11 bug was “Possible null pointer dereference of createLog in getFileSK(int, int, String) on exception path”. This was also a genuine correctness error that was also fixed cloning the initialisation of createLog into the catch statement.

1 of the rank 15 bugs is “A prepared statement is generated from a nonconstant String in getFileLog(int)” However, that string must be an int (it is implicitly cast to string), and it is merely used to decide the file name that is being output. Therefore, this is a safe thing to do.

The rank 14 bug and the the other rank 15 bug are “Random object created and used only once in getRandomBytes(int)” and “Dead store to random2 in setUp()”, which is part of the code that we use to test our encryption. There is only one use of a random object outside of the actual code, so it is correct that we only generate it once. Furthermore, this does not affect the correctness of our code.

All other bugs are rank 16 and below. A lot of them pertain to testing but do not affect the correctness of our code or the test.

We have 69 bad practice bugs, most of which pertain to closing streams or PreparedStatements. Some of these are not actually bugs - we want to continually listen to a stream. Some of them will be corrected by the next milestone as they are merely performance impacting bugs. We have another 26 experimental bugs that are in the same vein.

We have 21 internationalisation bugs. However, the data we send is encoding independent as we merely want to send the byte stream that has already been encrypted.

We have 20 performance bugs, 5 of which are unread fields that we have yet to fully refactor out, 4 of which are concatenating strings through + and not stringbuffer (all in testing), and 11 about whether exceptions should be declared as static. The exceptions should not be declared as static because they are independent of the class they are defined in, and we did not want to create 11 extra files to hold the 11 extra exceptions we define.

There are 81 dodgy code bugs, half of which are an artifact of JavaFX. For example, things are not initialised in the constructor of a JavaFX controller, but rather through a separate method that is called *after* initialise() has been called, because initialise takes no arguments but is the method that called on construction by JavaFX. The other half are bugs such as redundant nullchecks and exception caught when exception not thrown, which are not correctness but rather performance bugs. As these are not correctness impacting bugs, we have decided to leave them to the next milestone to clean up.