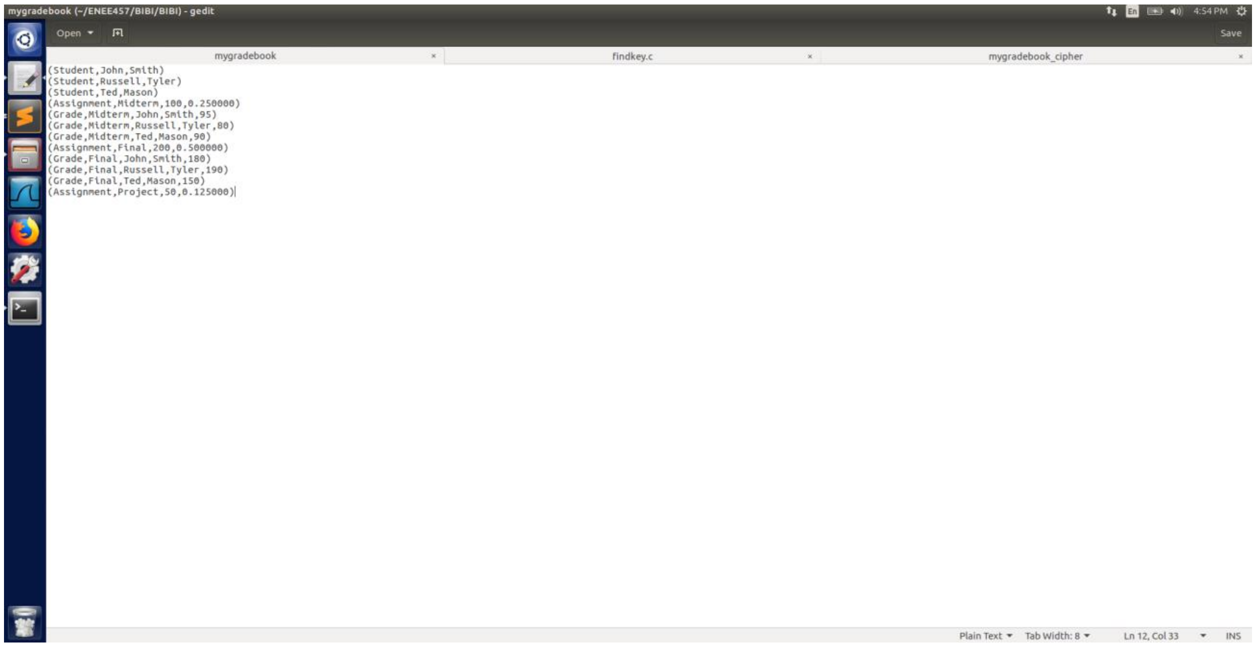
[redacted]

Build-it Project

1. Implementation design.
a. Gradebook Design:

For the gradebook file, every time the gradebook is add via the commands of gradebookadd, a tuple will be added to the gradebook with data respective to the options. Here is an example of how the gradebookadd options will change the gradebook:

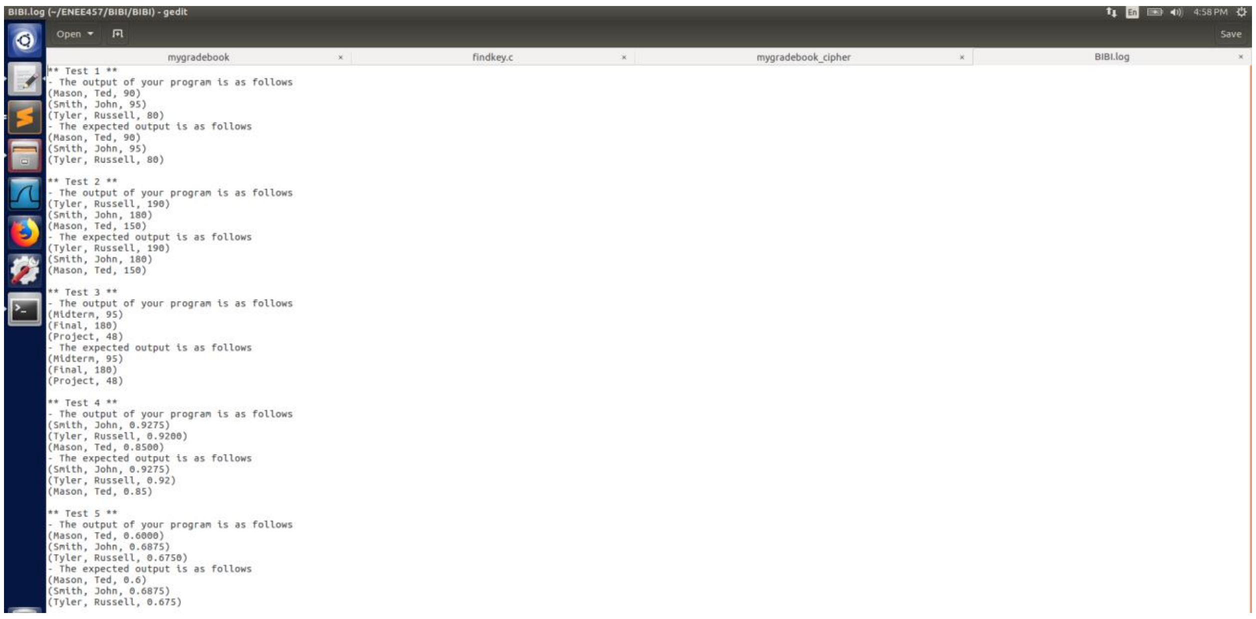


```
(Student,John,Smith)
(Student,Russell,Tyler)
(Student,Ted,Mason)
(Assignment,Midterm,100,0.250000)
(Grade,Midterm,John,Smith,95)
(Grade,Midterm,Russell,Tyler,80)
(Grade,Midterm,Ted,Mason,90)
(Assignment,Final,200,0.500000)
(Grade,Final,John,Smith,180)
(Grade,Final,Russell,Tyler,190)
(Grade,Final,Ted,Mason,150)
(Assignment,Project,50,0.125000)
```

The figure above shows all the possible tuple the gradebook will have:

- (Student, FirstName, LastName) → Represent a student with specified first name and last name
- (Assignment, AssignmentName, TotalPoints, Weight) → represent an assignment with the specified info
- (Grade, AssignmentName, FirstName, LastName, PointEarned) → Represent a grade entry corresponding to a student and an assignment.

From the 3 types of tuple above, I could query the gradebook to display the information of the students accordingly. Here is an example testing script result:



```
** Test 1 **
- The output of your program is as follows
(Mason, Ted, 90)
(Smith, John, 95)
(Tyler, Russell, 80)
- The expected output is as follows
(Mason, Ted, 90)
(Smith, John, 95)
(Tyler, Russell, 80)

** Test 2 **
- The output of your program is as follows
(Tyler, Russell, 190)
(Smith, John, 180)
(Mason, Ted, 150)
- The expected output is as follows
(Tyler, Russell, 190)
(Smith, John, 180)
(Mason, Ted, 150)

** Test 3 **
- The output of your program is as follows
(Midterm, 95)
(Final, 180)
(Project, 48)
- The expected output is as follows
(Midterm, 95)
(Final, 180)
(Project, 48)

** Test 4 **
- The output of your program is as follows
(Smith, John, 0.9275)
(Tyler, Russell, 0.9200)
(Mason, Ted, 0.8500)
- The expected output is as follows
(Smith, John, 0.9275)
(Tyler, Russell, 0.92)
(Mason, Ted, 0.85)

** Test 5 **
- The output of your program is as follows
(Mason, Ted, 0.6000)
(Smith, John, 0.6875)
(Tyler, Russell, 0.6750)
- The expected output is as follows
(Mason, Ted, 0.6)
(Smith, John, 0.6875)
(Tyler, Russell, 0.675)
```

From the result above, the actual output matches the expected output from the SPEC.

However, all this is done on the plaintext file.

b. Security Design:

For the security design, I will be using AES-128-CBC. In the setup file, a key will be generated from u/random. The key is then used to calculate a HMAC value. The HMAC value is then compared when the correct key is inputted through the command options. The program proceeds when the values match.

Gradebookadd will first decrypt the file with the given key and write the plaintext content of the file into a file of the same name, while keeping the HMAC value for subsequent calls. The old, encrypted file will be deleted. The program will begin to work on the plaintext file as if no encryption were in place. After the result of the command options has been written back to the file. The content of the file is then encrypted using the same key, along with a randomly generated IV (again through u/random). The HMAC value is then stored in the new file for subsequent use.

Gradebookdisplay will use the key to decrypt the file and store the contents of the file into a new file. The program will then work on the new plaintext file as if no encryption were in place. After finished displaying the requested data, the program will delete the newly created file.

I was not able to implement the security aspect of the programs, since I could not decrypt what I encrypted with the openssl library. I suspected that there could be problems with how I parsed the key and iv that were embedded along with the ciphertext.

2. Possible attacks and countermeasures:
    a. Buffer overflow attack.

I first implemented the buffers within the file as fixed size, which are vulnerable to bufferoverflow attacks. The attacker could input a name of longer strings and it could crash the program with a segmentation fault. A countermeasure that I implemented was to check the size of the input before using the input as a parameter for the programs. If longer inputs were provided, the program will throw an error (return 255 and prints invalid).

b. Middle-man attack

From the SPEPC, the adversary who corrupted the server but does not know the key could not make queries as well as modify the gradebook with gradebookdisplay and gradebookadd. With the security model described above, the middleman will not be able to interpret data from the gradebook. However, I could not find a way to prevent the adversary from changing the data from the file to make the gradebook a corrupted one. I changed the method of encryption to AES-OFB instead of CBC. From project 3, we learned that modes of operation affect the ability to recover data tremendously, and that OFB is the mode that can recover the most data from a corrupted cipher text. Therefore, using this mode will partially prevent the adversary from corrupting the data.

c. The adversary could effectively launch a script through inputs of the program.

The program specifications would prevent this as the names (of student, files, and assignments) are alpha-numerics, meaning that it only contains numbers, upper and lowercase alphabetic characters. This prevents the adversary from executing scripts from the input. I have not implemented this since I did not have enough time. However, as mentioned above, I have restricted the length of input to prevent more harmful attacks that requires longer script. I also formatted the file and the buffer with comma separated strings, so that scripts are not as easy to launch from inputs alone.

d. Chosen Ciphertext attack and Chosen Plaintext attack.

The program is not CPA and CCA secured, because the HMAC as described above, is deterministic so that we can compute and compare for every subsequent usage of the programs. I would have implemented the countermeasure by encrypt and then authenticate instead of authenticating and then encrypt. However, I am not sure how to accomplish this with the library as well as with how the specs are specified. I have partially tried to implement countermeasure for the CPA attack by randomizing the IV for each encrypt. However, as mentioned above, I was not able to decrypt the content of the file from the encrypted file, thus was not able to test this vulnerability.