[redacted]

Design Document

General Information

The program can be compiled by typing "make" as long as all files (Makefile, gradebookadd.c, gradebookdisplay.c, my_data.h, crypto.c, and setup.c) are in the same directory. The user can then type "setup", "gradebookadd", or "gradebookdisplay" with the the associated necessary arguments to run the program. To recompile, the user has the option of typing "make clean" to completely start over.

Setup.c

Setup.c takes as input the -N option and the chosen filename of the user. Each user input was checked to ensure that the expected values were given to the program as discussed in items 1 and 2 in the Vulnerabilities section below. The program generates a key that is shown in STDOUT. This key must be used to manipulate and view data via gradebookadd.c and gradebookdisplay.c. This key was generated using RAND_bytes since it fills a buffer with cryptographically strong pseudo-random bytes. This was used rather than a general random function since it is more secure. An IV was also generated using RAND_bytes.

Additionally, an encrypted gradebook and a decrypted gradebook are created and initialized in setup.c. The same encryption and decryption, AES 256 CBC, was utilized for all three programs (setup.c, gradebookadd.c, and gradebookdisplay.c). The encryption and decryption were done via the encryption and decryption functions defined in crypto.c. The encrypted gradebook's fields are the IV, the encrypted data, and the MAC. The decrypted gradebook's fields are the gradebook name, the current number of students, the current number of assignments, an array of student structs, and an array of assignment structs. Each student struct consists of the first name, last name, and an array of grades for the student. Each assignment struct consists of the assignment name, the points associated with that assignment, and the assignment weight. The MAC for the encrypted gradebook described above was created via two steps. I first did a SHA256 hash on the key generated from RAND_bytes. This created the MAC tag. I then utilized the HMAC function to create the MAC by using the tag and encrypted gradebook. The contents of this encrypted gradebook is then written to the chosen gradebook file.

Gradebookadd.c

For gradebookadd.c, the inputs are checked to ensure that the expected value was inputted. All inputs were checked using the methods described by items 1 and 2 in the vulnerabilities section. Additionally, the program checks that two commands are not inputted simultaneously (an example would be -AS and -AG). The program also checks that extra arguments were not inputted and checks that each command is valid. Furthermore, item 4 in the vulnerabilities section was utilized to make sure that the gradebook file was not altered. If it is not altered, the encrypted gradebook is decrypted, and, if all following checks are passed, the gradebook is legally changed. If the gradebook is legally changed, this content is encrypted again, and the MAC is updated as described in item 4 under vulnerabilities. The contents of this encrypted gradebook are then written to the chosen gradebook file.

As far as the general function for each instruction, a structure was utilized to store the information related to each input (such as the filename, the assignment name, associated points, grades, etc). If the previously mentioned checks are clear, then this structure's fields are populated, and the action (such as add assignment) begins. For the chosen instruction, another series of tests begins depending on the chosen action. For instance, the existence of the particular assignment would be checked before any action was taken for delete assignment. For each function, once the user has been verified, a pointer to the decrypted gradebook is supplied as input to the associated function. Then the necessary parameter like the current number of students or the number of current assignments is utilized to search for and manipulate the desired data.

Gradebookdisplay.c

Gradebookdisplay.c utilizes a similar process as gradebookadd.c. It first checks the size of all user input as described by item 2 in the vulnerabilities section below. It also checks each input to ensure that the expected type was given by the user (item 1 under vulnerabilities). It also checks that more than one command has not been specified. The user's key is then verified, and the encrypted gradebook is checked to make sure that it was not altered via the HMAC function (using the method described in item 4 under vulnerabilities). If the MAC is correct, the encrypted gradebook is decrypted, and the desired action begins. If it is not correct, an error occurs, and the program exits.

Subsequent checks occur to make sure that the desired command is valid. For example, when doing print student, it makes sure that the student exists before attempting to print the content. A structure that consists of the relevant fields (like the student name, assignment name, etc) is passed to each function as well as the decrypted gradebook. The desired information is displayed if all checks are passed.

General Notes:

- Note that for setup.c when a gradebook is successfully created, an additional print statement is shown saying that the gradebook was successfully created.
- For gradebookdisplay.c, note that an edge case was considered where the number of points is zero. (Number of points in the assignment prompt on the course website was described as "non-negative" and not explicitly positive). In this case where the points are zero, an error is outputted in the final grade calculation. This is because, based on the sample input and output, the grade that a student received is divided by the assignment points. To avoid a potential break due to dividing by zero, "invalid" is printed, and the program is returned.
- For gradebookdisplay.c, also note that when an assignment is deleted via gradebookadd, the associated student grades are deleted. This affects gradebookdisplay.c because it alters the final grade calculation in the -PF option.
- Note that if user input exceeds the given length of any parameter given in my_data.h, "invalid" is printed. For instance, if the inputted first name exceeds the maximum number of characters for the first name specified in the my_data.h file, then "invalid" is printed. This occurs for filenames, assignment names, and first and last names. Additionally, if the number of inputted assignments exceeds the maximum number of assignments, "invalid" is printed. This will also occur if someone tries to add a student and the resulting number of students is greater than the maximum number of students.
- Note that if the encrypted gradebook MAC and the MAC acquired from the user's key don't match (i.e. the key is incorrect or the encrypted gradebook has been altered), "invalid" is printed, and the program returns 255.

- Note that all grades are initialized to zero when a student is added and no grade has been added yet.

General Assumptions:

- The key size, MAC size, and IV size were chosen to be 32 bytes as specified in the my_data.h file. Additionally, the maximum number of students is 400. The maximum number of assignments is 100.
- The maximum number of characters for an assignment name, first name, or last name, is 30.
- If any of these values are exceeded, "invalid" will be printed and the program will return 255.

Vulnerabilities:

1. The program uses blacklisting and whitelisting to check for malformed input. It includes making sure that the filename only consists of alphanumeric characters. Additionally, it makes sure that the key only consists of hex digits. For assignment names, it ensures that the input only consists of characters 0-9, a-z, or A-Z. For the number of points, the program checks that the value is a non-negative integer. For the weight, the program checks that the value is a real number between [0,1]. For first and last names, the program checks that the inputted name is 0-9, a-z, or A-Z. For the grade, the program checks that the inputted number is a non-negative integer.
    a. Relevant lines of code in setup.c:
        i. 73-84
    b. Relevant lines of code in gradebookadd.c:
        i. 267-278, 299-306, 361-367, 383-389, 404-421, 464-470, 509-515, 533-539, 574-579, 598-604, 640-644, 663-669, 687-693, 708-714
    c. Relevant lines of code in gradebookdisplay.c:
        i. 289-299, 321-328, 377-383, 420-426, 440-446
2. The program also avoids buffer overflows by checking that all user input does not exceed the specified lengths in the my_data.h file.
    a. Relevant lines of code in setup.c:
        i. 67-72
    b. Relevant lines of code in gradebookadd.c:
        i. 261-266, 293-298, 355-359, 458-462, 503-507, 527-531, 568-572, 592-596, 633-637, 657-661, 681-685
    c. Relevant lines of code in gradebookdisplay.c:
        i. 370-375, 414-418, 433-438
3. The program uses encryption to ensure that only individuals with the correct key can view or alter the associated gradebook.
    a. Relevant lines of code setup.c:
        i. 129
    b. Relevant lines of code in gradebookadd.c:
        i. 835
    c. Also see crypto.c file.
4. The program utilizes a MAC to ensure that users cannot illegally change the ciphertext file. First, the inputted key is hashed to create the MAC tag. This tag is then inputted into the HMAC function along with the encrypted gradebook. If the resulting MAC matches the MAC of the

encrypted gradebook, then the file was not manipulated. If the gradebook is manipulated legally via gradebookadd.c, a new MAC is generated, and the MAC field of the encrypted gradebook is updated.

   a. Relevant lines of code in setup.c:
      i. 131
   b. Relevant lines of code in gradebookadd.c:
      i. 777-782, 836
   c. Relevant lines of code in gradebookdisplay.c:
      i. 505-509

5. The key was generated using RAND_bytes rather than seed to ensure that the key is very difficult to guess. (So that it consists of cryptographically strong pseudo-random bytes).
   a. Relevant lines of code in setup.c:
      i. 120
6. Additionally, encrypt then authenticate was utilized rather than authenticate then encrypt and other methods to ensure that my algorithm is secure.
   a. Relevant lines of code in setup.c:
      i. 129-131
   b. Relevant lines of code in gradebookadd.c:
      i. 835-836