Design Document

Describe your overall system design in sufficient detail for a reader to understand your approach without reading the source code directly. This must include a description of the format of your gradebook.

After all executables are compiled by typing "make" in the cmdline, the user should first create a new gradebook via

```
./setup -N <gradebook name>
```

Once setup is called, a randomly generated key will be printed to stdout – this is the key that must be used to access and modify contents in the gradebook (via `gradebookadd` and `gradebookdisplay` executables). If the user tries to create a gradebook with an already existing gradebook name, setup will print "Invalid" and exit. If the user tries to access a non-existing gradebook through `gradebookadd` or `gradebookdisplay`, "Invalid" will be printed to stdout and the program will exit. To modify or display contents in the gradebook, the user can use the `gradebookadd` and `gradebookdisplay` executables. The commands must have the beginning format of either:

```
./gradebookadd -N <name> -K <keyGivenFromSetup> <Action> …
./gradebookdisplay -N <name> -K <keyGivenFromSetup> <Action> …
```

Format of the Gradebook and Data (see data.h for more details)

I defined the following constants to limit the length of names and the max # of students and assignments a gradebook can have:

- MAX_NAME_LEN = 100
- MAX_STUDENTS = 1000
- MAX_ASSIGNMENTS = 100
- IV_SIZE = 32
- MAC_SIZE = 32

Throughout my code, I check if the input doesn't exceed the limitations above. For ex: before adding a new student, I make sure the name length doesn't exceed MAX_NAME_LEN chars and the gradebook has less than 1000 students.

I have an EncryptedDatabook structure that stores the iv, encrypted data (stores the encrypted version of the gradebook), and mactag. The iv and mactag are used for encryption and authentication purposes, which are discussed further below.

I have a DecryptedGradebook structure template that stores the following:

- Gradebook name
- Total number of students
- An array of Student structures
- Total number of assignments
- An array of Assignment structures
- Total sum of all the assignment weights

I use a PaddedDecryptedGradebook, a structure that has the same fields as DecryptedGradebook, but it has one additional field: padding. Since the encryption/decryption algorithms require passed in data to be a multiple of 64, I added padding to make the structure size a multiple of 64 so encryption/decryption will work properly. I had to create a DecryptedGradebook structure as a baseline size template for the PaddedDecryptedGradebook, as I use the size of the DecryptedGradebook to calculate how much padding I need for the PaddedDecryptedGradebook structure:

```
char padding[64 - (sizeof(DecryptedGradebook) % 64)];
```

The PaddedDecryptedGradebook is the gradebook that the professor/TA's interact with when modifying the gradebook. Besides the padding field, I also store the gradebook name, number of students and assignments in the gradebook, the total weights of all assignments in the gradebook, and 2 arrays of structures:

- **Array of Student structures,** where each Student structure consists of the first name, last name, an array of grades for that student, and the final grade.

  The size of the array of grades is the number of assignments in the gradebook. Each index corresponds to an assignment. Ex: if the number of assignments = 3, then grades[0] is the grade for the 1st assignment, grades[1] is the grade for the 2nd assignment, and grades[2] is the grade for the 3rd assignment

- **Array of Assignment structures,** where each Assignment structure consists the assignment name, number of points of the assignment, and the assignment weight

After the initial setup is called, the initially created gradebook is encrypted and stored to a file. The encrypted contents include the IV and encrypted gradebook data. The EncryptedDatabook structure also uses HMAC to produce a tag that will be used to ensure authenticity and integrity – thus only allowed users with the correct key can make changes and view contents of the gradebook. To use the HMAC, the inputted key (same as the randomly generated key from calling setup) is first hashed via SHA256 – producing a mac_key. HMAC is then run on the mac_key and encrypted databook/IV (these two fields of EncryptedGradebook) to produce the mac_tag – this is also stored (but not encrypted!) in the EncryptedGradebook structure. The mac_tag will later be used when a user tries to access the gradebook.

Additionally, the random key is given to the user and changes can now be made via `gradebookadd` and the user can view contents via `gradebookdisplay`.

Upon calling `gradebookadd` or `gradebookdisplay`, the program first takes in the command line arguments and parses them. If any illegal arguments are received, the program prints "invalid" and exits with error code 255.

Example illegal arguments include:
- No lastname provided after "-LN"
- Inputting an integer when it was expecting a float

After parsing through the given arguments, the program then checks if the passed in gradebook name file exists; if it does not then the program prints "invalid" and exits with error code 255.

If the file exists, the program reads in the encrypted data and stores it in an EncryptedGradebook structure variable. The program then uses HMAC to verify that the user inputted the correct key and the gradebook wasn't modified by an unauthorized user:

1. the mac_key used in the HMAC is generated by performing the SHA256 hash on the inputted key.
2. The HMAC is then run on the produced mac_key and the EncryptedDatabook Structure (IV and encrypted_data fields) to produce a temp_mac_tag.
3. The program then checks if the temp_mac_tag and the stored mac_tag are the same – if they are, then the correct key was inputted and no unauthorized changes were made to the gradebook
4. Otherwise, if the temp_mac_tag and the stored mac_tag are NOT the same, the program prints "Invalid" and exits.

Once the tag is verified, the gradebook is decrypted – now the user can make changes and view contents of the gradebook. If any illegal arguments are passed in when calling `gradebookadd` or `gradebookdisplay` the program will print "Invalid" and exit.

Once changes are made to the gradebook the encryption algorithm is then called to encrypt the updated gradebook. Additionally, a new IV and mac_tag are generated and stored in the Encrypted gradebook that's being written to the file – these are used to indicate that the gradebook was changed by an authorized user.

List four specific attacks you have considered, and describe how your implementation counters these threats. (Please note the relevant lines of code for your defense.)

ATTACKS/COUNTERMEASURES:

1. **Preventing the attacker from obtaining the key:**
   When producing a random key or iv, it is important to use a secure random generator. I used the method RAND_bytes( ). This is more secure than using a random generator that takes in a seed – otherwise the attacker can learn the pattern of the random keys generated and gain access to the gradebook.

   For code references:
   - setup.c: line 71, 89
   - gradebookadd.c: line 642

2. **Prevent unauthorized viewing of the gradebook:**
   To prevent the gradebook from being viewable by anyone, the gradebook must be stored in an encrypted format in the file on the disk. Thus I used the secure encryption and decryption algorithms from the openssl library. Decryption was only called after verifying that the user inputted the correct key and no unauthorized changes were made (see #3 below). Encryption was called after setup was called and every time changes were made to the gradebook.

   For code references:
   - setup.c: lines 96-99
   - gradebookadd.c: lines 612-615, 644-645
   - gradebookdisplay.c: lines 449-450, 473-476

3. **Verifying command line arguments:**
   To avoid illegal arguments, blacklisting and whitelisting were used to avoid any unwanted inputs and verify that the command line inputs were valid. These methods were used when parsing the input commands (before the gradebook is modified or displayed). Some example checks done in the program are explained below:

   *note: the line numbers provided are just a couple instances of where the checks occur, more instances occur in other parts of the program as well.

   i. **names do not contain any unwanted characters**
      1. make sure gradebook name only contains alphanumeric characters including underscores ("_") and periods. all other characters are considered invalid
         (setup.c: lines 46-64)

      2. check if names contain only alpha chars or alphanumeric chars
         (gradebookadd.c: lines 170-172, 254-256)
         (gradebookdisplay.c: lines 149-152, 195-197)

ii. **received a float number when expecting a float**
(gradebookadd.c: lines 43-60, 199-202)

iii. **received a non-negative integer**
(gradebookadd.c: lines 189-194)

iv. **Make sure exactly one action is specified in the command**
(gradebookadd.c: lines 118-141)
(gradebookdisplay.c: lines 97-114)

v. **Make sure valid arguments are given based on the input**
Ex: if given "-FN", then the next argument should be a valid name containing only alpha characters
(gradebookadd.c: lines 293-368)
(gradebookdisplay.c: lines 137-161)

vi. **the first few required arguments are correct in the following format:**

```
./gradebookadd -N <name> -K <keyGivenFromSetup> <Action> …
./gradebookdisplay -N <name> -K <keyGivenFromSetup> <Action> …
```

The program will print Invalid if:
- the 2nd argument is not "-N"
- the 3rd argument contains unallowed characters
- the 4th argument is not "-K"
- the 5th argument is not the correct key
- the 6th argument is not one of the valid actions

(gradebookadd.c: lines 141, 588-608)
(gradebookdisplay.c: lines 114, 449-469)

vii. **In gradebook display, make sure that exactly one instance of '-A' or '-G' is provided**
(gradebookdisplay.c: lines 163-179)

viii. **When calling setup, make sure the file does not already exist. When calling gradebookadd or gradebookdisplay, maek sure the file exists**
(setup.c: lines 32-37)
(gradebookadd.c: lines 581-586)
(gradebookdisplay.c: lines 442-447)

4. **Prevent an overflow (Ex: buffer and heap overflow)**
   Before making a new gradebook, I make sure the length of the gradebook name does not exceed the max length allowed.
   (setup.c lines 39-42)

   Before modifying the gradebook, I make sure the total number of students or assignments are not at max capacity and the names of students/assignments do not exceed the max length allowed.
   (gradebookadd.c lines 298, 316, 383, 470)

5. **Verifying correct keys and no unauthorized changes were made**
   HMAC was used to ensure the correct key was inputted and no unauthorized changes were made to the gradebook. Some details were mentioned above while discussing the implementation and design of the system:

   After calling the encryption algorithm (which encrypts the IV and encrypted gradebook data), the inputted key (same as the randomly generated key from calling setup) is first hashed via SHA256 – producing a mac_key. HMAC is then run on the mac_key and encrypted databook/IV (fields of EncryptedGradebook) to produce mac_tag – this is also stored (but not encrypted!) in the EncryptedGradebook structure. The mac_tag is then used when the user tries to access the gradebook:

   The mac_key used in the HMAC is generated by performing the SHA256 hash on the inputted key. The HMAC is then run on the produced mac_key and the EncryptedDatabook Structure (IV and encrypted_data fields) to produce a temp_mac_tag. The program then checks if the temp_mac_tag and the stored mac_tag are the same – If the temp_mac_tag and the stored mac_tag are NOT the same, the program prints "Invalid" and exits. Otherwise, we know the correct key was inputted and no unauthorized changes were made to the gradebook. Once these are verified, the user can then modify and view the gradebook.

- setup.c
  - lines 88-109: initial encryption and mac_tag generation along with writing the encrypted gradebook to the file

- gradebookadd.c
  - lines: 588-608: reads in the encrypted gradebook and uses HMAC to verify that tags match – ensuring authenticity and integrity

- gradebookdisplay.c
  - lines: 449-469: reads in the encrypted gradebook and uses HMAC to verify that tags match – ensuring authenticity and integrity