




## ENEE457 Project 4 Secure Gradebook



### Part 1: Setup

Within  there should be three folders named “pretty\_table\_with\_debug”, “tuples\_with\_debug”, and “tuples\_without\_debug”.

- “pretty\_table\_with\_debug” will print the gradebook out in a formatted table. If you fail to run a command, the program will tell you what to fix. Please work with this folder if you are running the program manually.
- “tuples\_with\_debug” will print data out in tuple form with no extra information – the same as the sample outputs posted on [https://user.eng.umd.edu/~danadach/Security\\_Fall\\_20/sample.html](https://user.eng.umd.edu/~danadach/Security_Fall_20/sample.html). If you fail to run a command, the program will tell you what to fix.
- “tuples\_without\_debug” will print data out in tuple form with no extra information – the same as the sample outputs posted on [https://user.eng.umd.edu/~danadach/Security\\_Fall\\_20/sample.html](https://user.eng.umd.edu/~danadach/Security_Fall_20/sample.html). If you fail to run a command, the program will only print “invalid”. Please perform automated correctness tests on this folder.

Within each folder, there should be 6 files: crypto.c, setup.c, gradebookadd.c, gradebookdisplay.c, data.h, and Makefile. You can run the makefile using “Make” to compile the programs. Afterwards, you should see 3 executables (in addition to the .o files): setup, gradebookadd, and gradebookdisplay. Simply run the three programs based on the documentation found at [https://user.eng.umd.edu/~danadach/Security\\_Fall\\_20/SPEC.html](https://user.eng.umd.edu/~danadach/Security_Fall_20/SPEC.html) along with the extra information found in part 2.

Before running the program, you may want to change the maximum number of students/assignments that can be placed within a single gradebook. You can do this under “data.h” lines 9-10. By default, max students = 500 and max assignments = 100. Higher numbers will require more storage space. If you change these numbers after creating your gradebook, you will no longer be able to edit/open older gradebooks. Please make sure you are satisfied with these numbers before you run “setup”.

If you want to delete the object files and executables, simply type “Make clean”. If you want to delete all .bin files from the directory, run “Make clean\_bin”. This may be useful if you name your gradebooks with .bin at the end (but you don’t have to).



## Part 2: Overall System Design

To run the program, please follow the documentation found at [https://user.eng.umd.edu/~danadach/Security\\_Fall\\_20/SPEC.html](https://user.eng.umd.edu/~danadach/Security_Fall_20/SPEC.html). In addition to those specifications, there are a few additional rules:

- The key must be 64 hexadecimal characters long (aka 32 bytes, or 256 bits). You will see the key after running “setup”.
- Assignment points/grade must be less than 1 million.
- An assignment’s max points must be greater than 0.
- You are allowed to give grades that are higher than assignment points (extra credit). Extra credit may allow final grades to be greater than 1.
- The size of all input strings must be less than or equal to 31 characters. For example, when inputting the assignment name using “-AN *assignment-name*”, *assignment-name* must be 31 characters or less.
- The number of command line input arguments must be less than or equal to 30.
- Grades will be initialized to 0 for a new assignment.
- When in doubt, the debug messages will usually give you feedback on what’s wrong with your input.

Gradebook Format (data.h):

- The gradebook is stored within nested structures.
- Each student has a structure containing their first name, last name, and grades. “grades” is an integer array where each index corresponds to an assignment.
- Each assignment has a structure containing the assignment name, max points, and weight of the assignment.
- The “DecryptedGradebook” structure contains an array of student structures along with an array of assignment structures. The index of the “assignments” array corresponds to the index of the students’ “grades” array.
  - For example, for the assignment located at index 4, there should be an assignment structure located at DecryptedGradebook.assignment[4] along with corresponding grade info at DecryptedGradebook.students[#].grades[4].
- “DecryptedGradebook” has 4 additional fields are used to manage student/assignment insertion/deletion: # of students, # of assignments, student\_index\_filled (a boolean array that decides which student array index is taken), and assignment\_index\_filled (a boolean array that decides which assignment array index is taken). Then, there’s 1 additional field called “padding” that will forever be 0 (null characters). “padding” makes DecryptedGradebook a multiple of 64 bytes to simplify encryption/decryption.
- Finally, the “EncryptedGradebook” structure contains iv, encrypted data, and mac. “iv” and “encrypted data” together makes up the “ciphertext” of the DecryptedGradebook. The “mac” field is a tag that enforces the integrity of the ciphertext. Once “EncryptedGradebook” is created, the contents may be dumped directly onto the disk/network/internet/cloud while preserving privacy and integrity.



An overview of setup.c:

- Setup creates a new gradebook and prints a 64-character long key. You should keep this key and use it when accessing the gradebook in the future.
- The newly created gradebook will be stored within the program as a “DecryptedGradebook” with all fields initialized to 0.
- The “DecryptedGradebook” will be encrypted, and the ciphertext will be tagged. The result will be written to a file specified by the input. See part 3 for further details on the encryption/MAC scheme.

An overview of gradebookadd.c:

- Gradebookadd will read the inputted file and place it into “EncryptedGradebook”. The encrypted gradebook will be verified for integrity/authenticity based on the input key, then decrypted into “DecryptedGradebook”. See part 3 for further details on the verification/decryption scheme.
- Important helper methods that modifies “DecryptedGradebook”:
  - parse\_cmdline(): Parses the command line input and makes sure the input is not malicious.
  - convert\_cmdline(): Converts command line inputs into useful data (such as integers, floats, etc). Stores the data in the structure “ExecutionData”. “ExecutionData” tells the program how to process the gradebook.
  - modify\_gradebook(): Changes “DecryptedGradebook” based on “ExecutionData”.
- Finally, “DecryptedGradebook” will be encrypted with a new randomly generated iv, and the ciphertext will be tagged using a MAC. The result will be written to a file specified by the input.

An overview of gradebookdisplay.c:

- Gradebookdisplay will read the inputted file and place it into “EncryptedGradebook”. The encrypted gradebook will be verified for integrity/authenticity based on the input key, then decrypted into “DecryptedGradebook”.
- The contents of “DecryptedGradebook” will be analyzed, and information will be printed based on user input. Important helper methods:
  - parse\_cmdline(): Parses the command line input and makes sure the input is not malicious. Place useful information into “ExecutionData” that tells the program how to process the gradebook.
  - p\_assignment(), p\_student(), p\_final(): Prints gradebook data based on what’s in “ExecutionData”. These methods will create an internal array that stores tuple structures. Each tuple will be printed using printf.



### Part 3: Defense and Mitigation Techniques

Note: All line numbers are based on the code found in the “pretty\_table\_with\_debug” folder. If use the code found in “tuples\_with\_debug” or “tuples\_without\_debug”, the line numbers for gradebookdisplay.c will be slightly different.

All methods are fully implemented in my code.

#### Method 1: Gradebook Privacy

- An attacker cannot retrieve any information about the gradebook without the 256-bit key. This is achieved by using proper encryption/decryption techniques.
- Additionally, even if a valid user encrypts the same gradebook twice (for example, the user adds the student John Smith, then deletes the student John Smith), an attacker cannot figure out whether those two gradebooks are the same. This is achieved by generating a random iv every time the user calls gradebookadd.
- Encryption procedure:
  - A cryptographically secure random iv will be generated for every encryption call (crypto.c lines 172, 147-155).
  - “DecryptedGradebook” will be encrypted with the random iv along with the 256-bit key using AES 256-bit CBC (crypto.c lines 179, 21-68).
  - The iv + encrypted data will be placed in “EncryptedGradebook”’s iv + encrypted\_data fields and together they’ll be known as the ciphertext (crypto.c lines 178-179).
- Decryption procedure:
  - Authentication/verification must be performed first before performing decryption (see Gradebook Integrity and Authenticity below).
  - The “EncryptedGradebook”’s iv + encrypted\_data along with the user-inputted 256-bit key will be fed through AES 256-bit CBC decryption (crypto.c lines 214, 71-119).
  - The result will be placed within “DecryptedGradebook”. Since “DecryptedGradebook” is a structure with a multiple of 64 bytes, we can place the result of the decryption directly into “DecryptedGradebook” with no buffer overflow. We are also allowed to assume that the decrypted data length is the same as the size of the DecryptedGradebook due to our authentication/verification procedure.

#### Method 2: Gradebook Integrity and Authenticity

- An attacker cannot modify any bits in the encrypted gradebook file without the program rejecting it. This is achieved by using proper MAC techniques. Furthermore, the attacker cannot add or delete ANY bits in the file, or the file reading procedure will reject the file as well.
- Additionally, the MAC tag will be secured by a different key (the hash of the master key). This will prevent any attacks due to potential weaknesses/similarities between the AES and HMAC procedures (even though there seems to be no known similarities as of today).
- Tagging procedure:
  - Gradebook encryption must be performed first before performing MAC (see Gradebook Privacy above).
  - A new HMAC key will be generated via a SHA256 of the encryption key (crypto.c lines 169, 123-155).



- After encryption, “EncryptedGradebook”’s iv + encrypted\_data fields will be passed into the OpenSSL’s HMAC procedure alongside the new HMAC key (crypto.c line 183).
- The new MAC tag will be placed into “EncryptedGradebook”’s mac field.
- Finally, the “EncryptedGradebook” will be allowed to be written directly to a file.
- Authentication/verification procedure:
  - When reading the file, the program will immediately abort if the file it reads is shorter than or longer than the size of the “EncryptedGradebook” structure (gradebookadd.c lines 717, 722-728, gradebookdisplay.c lines 635, 640-646). If the sizes are equal, the program will place the file’s data into “EncryptedGradebook”.
  - A new HMAC key will be generated via a SHA256 of the encryption key (crypto.c lines 200, 123-155).
  - The “EncryptedGradebook”’s iv + encrypted\_data fields will be passed into the OpenSSL’s HMAC procedure alongside the new HMAC key (crypto.c line 203).
  - The resulting MAC tag will be compared with the old MAC tag contained in “EncryptedGradebook.mac”. If they are the same, the verification will successfully complete. If they are different, verification will return INVALID and the program will abort.

### Method 3: Preventing buffer overflows

- The program prevents users from overflowing internal buffers (aka a buffer overflow attack). Depending on operating-system level defense, attackers may be able to:
  - Force the program to read/print out extra data within the program that should have been private.
  - Insert malicious code on the stack that the program will eventually run.
  - Insert malicious code on the stack that results in a malicious procedure through return-oriented programming.
- These attacks were prevented by checking the sizes of the user inputs. All user inputs will be checked using strlen(). The length of each input must be less than 32 characters so that they will fit in the buffers (gradebookadd.c lines 78-414, gradebookdisplay.c lines 77-283).
- strlen() is capped at 32 characters as well to prevent an input without a null character from causing strlen() to calculate the length further into memory.
- When copying user input into internal buffers, strncpy() is used instead of strcpy(). I can force strncpy() to copy a maximum of 31 characters so that buffer overflow never occurs even if I miss a check with strlen().
- Mentioned within authentication/verification procedure as well: When reading the file, the program will immediately abort if the file it reads is shorter than or longer than the size of the “EncryptedGradebook” structure (gradebookadd.c lines 717, 722-728, gradebookdisplay.c lines 635, 640-646). This will prevent buffer overflows where the attacker adds bits to the file.
- Mentioned within gradebook format (data.h): A field named “padding” is added to the end of the “DecryptedGradebook” to ensure that the structure can hold everything provided by the decryption algorithm. This is achieved by creating another structure named “CalcGradebookSize”, which contains the same fields as “DecryptedGradebook” (but without the padding). sizeof(CalcGradebookSize) will be the size of DecryptedGradebook’s fields without padding, so the size of the padding can be calculated as “64 - (sizeof(CalcGradebookSize) % 64)”



bytes to ensure that the size of DecryptedGradebook is exactly a multiple of 64. This will let me avoid automatic padding by the encryption/decryption formulas.

#### Method 4: Whitelisting and blacklisting user inputs

- The program aggressively combines whitelisting with blacklisting in order to parse user inputs. A malicious user will not be able to inject malicious code or input anything without null-terminating bytes. Proper whitelisting/blacklisting will ensure that the program's operation is controlled by the programmer (me), instead of performing unintended operations. It will also be difficult for an attacker to crash my program if all inputs to the program are properly sanitized (gradebookadd.c lines 78-414, gradebookdisplay.c lines 77-283).
- Flags such as "-N" must be the correct length (done using strlen()), which implies a null-terminating byte at the correct location.
- After checking for length, flags must be located at the correct array index within argv (done using strncmp() on argv). Notably, I always perform strlen() first in order to avoid strncmp() comparing further than intended, just in case a user inputs an argument without a null-terminating byte.
- Additional checks include (but not limited to):
  - Parsing user inputs to only allow alphanumeric/alphabetic/numeric/hexadecimal characters depending on the flag.
  - Only 1 decimal allowed in *add-assignment's assignment-weight* field to avoid faulty strtod() (string to float).
  - Numerical inputs cannot be too large to avoid integer overflow and faulty strtoul() (string to long).
  - Assignment max grade must be > 0 to avoid division by 0 errors.
  - Number of command line arguments must be less than or equal to 30.
  - Key must be 64 bytes long (program will then convert it into 32 bytes/256 bits); no more, no less.
  - There must always be a valid input after a flag (if the flag requests a string/number) instead of another flag.
  - All required flags (along with their corresponding input) must be found in the correct order; missing a required flag will not be accepted.

#### Method 5: No function pointers

- I was contemplating using function pointers for my sorting algorithm so that I can use a single bubble sort algorithm function. The bubble sort algorithm will be able to sort based on either character arrays (strings) or numbers depending on which comparison function is used (via a function pointer). I decided against this for the sake of security.
- If function pointers are available, there is a risk of an attacker performing buffer overflow to override the function pointer. The function pointer will then point to wherever the attacker desires. This is an unacceptable vulnerability.
- There are no function pointers in my code. Even if an attacker manages get around my strlen checks and overflow parts of the buffer, they will not be able to execute anything they want using function pointers.