Design Document

Data Structure

Each program uses the same Gradebook data structure. The Gradebook struct holds a Student pointer, Assignment pointer, and each of their array lengths. The Student struct holds a char pointer for first name and last name. The Assignment struct hold the assignment name, points, weight, grade pointer, length of grades array, and the number of characters in the weight value as a string.

Whenever an array needs to increase or decrease in size, the programs use realloc to dynamically assign memory. The grades stored in each assignment and connected with their student because they are accessed by the same index in the Gradebook's student array. These arrays are always kept the same length and any grade that has not yet been assigned is stored as -1.

The setup program initialized an empty gradebook and the gradebookadd and gradebookdisplay programs decrypt and parse the file into this gradebook structure.

```
typedef struct _Student {
    unsigned char *fname;
    unsigned char *lname;
} Student;

typedef struct _Assignment {
    unsigned char *assignment_name;
    int points;
    float weight;
    int weight_char_count;
    int *grades;
    int grades_length;
} Assignment;

typedef struct _Gradebook {
    Student *students;
    int students_length;
    Assignment *assignments;
    int assignments_length;
} Gradebook;
```

Buffer Format

The gradebook file is written as a single encrypted string. The buffer struct stores this string unencrypted. Whenever I read from the gradebook file I first decrypt and store the plaintext in the buffer, and whenever I write to the file, I convert the gradebook to a string and encrypt it.

The file is parsed with the following format: the first line contains an integer n for number of assignments, the second line contains an integer m for number of students. The following n lines hold the assignments formatted: assignment_name|points|weight|student1_grade|student2_grade|…\n
The following m lines hold the students formatted: student1_name|student1_lname|\n
Again, any grade that has not been assigned is stored as -1.

The gradebookadd and gradebookdisplay programs parse this buffer into a gradebook structure. The gradebookadd and setup programs convert a gradebook structure into this buffer.

Example unencrypted buffer:

```
2
3
midterm|100|0.250000|90|75|-1|
final|200|0.750000|175|150|-1|
John|Smith|
Russel|Tyler|
Ted|Mason|
```

Buffer data structure:

```
typedef struct _Buffer {
    unsigned char *Buf;
    unsigned long Length;
} Buffer;
```

Reading from file:

```
Buffer B = read_from_path(R.gb_name, R.key);
Gradebook *GB = (Gradebook *)malloc(sizeof(Gradebook));
get_Gradebook(GB, &B);
```

Writing to file:

```
Buffer P = print_Gradebook_to_buffer(GB);
write_to_path(R.gb_name, &P, R.key);
```

Encryption

My programs use AES-CBC 128-bit encryption. The key is generated by the setup program using 16 random bytes. When the key is read in by the other programs as a string of 32 hex characters, it is converted back to its original 16 bytes before decryption.

Before encrypting, my programs generate a 16 byte IV. After the ciphertext is generated the IV is appended to the end. When decrypting, the IV is removed from the ciphertext and used to decrypt.

Decoding key string:

```
void decode_hex_key(unsigned char *key_out, unsigned char *k, int k_len) {
    unsigned int msb, lsb;
    int i = 0;
    int j = 0;
    while (i < k_len) {
        msb = k[i] > '9' ? k[i] - 'a' + 10 : k[i] - '0';
        lsb = k[i+1] > '9' ? k[i+1] - 'a' + 10 : k[i+1] - '0';

        key_out[j] = (msb << 4) | lsb;
        i += 2;
        j++;
    }
}
```

Encryption:

```
EVP_CIPHER_CTX *ctx;
int len;
int ciphertext_len;

if (!(ctx = EVP_CIPHER_CTX_new()))
    handleErrors();

if (1 != EVP_EncryptInit_ex(ctx, EVP_aes_128_cbc(), NULL, key_data, iv))
    handleErrors();

if (1 != EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, plaintext_len
    handleErrors();
ciphertext_len = len;

if (1 != EVP_EncryptFinal_ex(ctx, ciphertext + len, &len))
    handleErrors();
ciphertext_len += len;

EVP_CIPHER_CTX_free(ctx);

for (int j=0; j < 16; j++) {
    ciphertext[ciphertext_len+j] = iv[j];
}
ciphertext_len += 16;
```

[redacted]

Security

I believe my programs are protected against buffer overflow and return-to-libc attacks as they use no fixed length buffers. Every time my buffer is written to or gradebook is read from, memory is allocated based on the length of the string. In the case where I am converting an integer to a string, I allocate memory based on the number of characters that converts too. For floats such as the assignment weight, I store the number of characters in the assignment when it is initially parsed as a string. Spatial safety is implemented by only allocating the exact amount of memory that is needed and temporal safety by freeing unused memory and preventing dangling pointers.

Reallocating buffer memory for an assignment:

```
for (int a_counter=0; a_counter < R->assignments_length; a_counter++) {
    Assignment a = R->assignments[a_counter];
    int an_len = strlen(a.assignment_name);
    int p_digits = a.points == 0 ? 1 : floor(log10(a.points)) + 1;
    unsigned char temp_str[an_len + p_digits + a.weight_char_count + 4];
    sprintf(temp_str, "%s|%d|%f|", a.assignment_name, a.points, a.weight)
    s = (unsigned char *)realloc(s, strlen(s) + strlen(temp_str) + 1);
    strcat(s, temp_str);
```

Because of the way my gradebook buffer is formatted, I am concerned about a potential integer overflow attack that could lead to my allocating less memory than is needed for my string, causing a buffer overflow. When I attempt to replicate this my program throws an error, but if I had more time, I would implement a manual check for this kind of behavior.

Attempts to modify the encrypted gradebook file result in an error being thrown by the OpenSSL decryption functions and invalid being printed. This protects me against chosen ciphertext attacks.

Result of valid command on an altered gradebook file:

```
[11/27/20]seed@VM:~/.../Starter_Code_export$ gradebookdisplay -N test -K
 0f1beccbe91494ede4fe5fc63971f2ff -PS -[redacted]
invalid
[11/27/20]seed@VM:~/.../Starter_Code_export$
```