WAYNE STATE UNIVERSITY

# Design and Implementation of A Secure Amazon S-3 Based File System

Muhtacin Manik

4 May 2020

CSC 4420

Professor Xu

Contents

## Introduction

For this final project, we are tasked to create a standalone RC4 that emulates openssl RC4. RC4 algorithm is one of the most used stream ciphers. RC4 encryption prevents data from being accessed without authorization. RC4 is used in many applications because of the speed and simplified implementation it provides. There were a few factors to consider when implementing this standalone. One of those factors was to recognize if the file would be salted or not. If a file is salted, it adds a 16-byte header containing a string "Salted__" and another random 8 bytes. Throughout this report, you will be able to see the difference between the implementation of salted files in encrypting/decrypting files using RC4.

Furthermore, we are tasked with making sure that all files using S3FS are automatically encrypted using rc4 with salt, and applications can transparently operate on the files without explicit decryptions. S3FS is a FUSE filesystem application backed by AWS simple cloud storage service that allows you to mount an Amazon S3 bucket as a local filesystem. Currently, S3FS is supported in Linux and macOS. When files are added to the local filesystem via the local bucket, they are automatically uploaded in Amazon S3 web client. Likewise, files uploaded directly to the web client are available via the local bucket if it is mounted.

This project was very rewarding in growing my knowledge in many aspects of this CSC 4420 class. I learned how to tackle many challenges while working with this project. In this report I will discuss my system information, project goals, tools and packages used, design, integration and implementation, future improvements, what helped, and a summary of what I have learned.

## Project Goals

There were several goals of this project with the last two being the main goals:

- To mount a bucket in S3 so it appears as a local directory on the Linux machine
- Any creation, deletion, or modifications of files in the local directory that is mounted is reflected in the corresponding S3 bucket
- Creating a stand-alone rc4, fully compatible with "openssl rc4", with both nosalt and salt options.This means that files encrypted/decrypted by the standalone should be able to be encrypted/decrypted by openssl rc4
- Integrate RC4 encryption and decryption with salt into S3FS. After integration, files uploaded directly into the bucket from the local directory should be encrypted with salt and be encrypted when viewing the file in Amazon S3 bucket. This means that files downloaded from the Amazon S3 bucket that are encrypted should also be able to become decrypted after being placed into the local directory mount and downloaded from Amazon S3.

## System Information

### OS Kernel/Distribution

- Kernel: Linux
- Kernel-Version: #56~20.04.1-Ubuntu SMP Mon Apr 12 21:46:35 UTC 2021
- Kernel Release: 5.8.0-50-generic
- Machine Hardware: x86_64

### CPU Information

| | |
|---|---|
| Architecture: | x86_64 |
| CPU op-mode(s): | 32-bit, 64-bit |
| Byte Order: | Little Endian |
| Address sizes: | 48 bits physical, 48 bits virtual |
| CPU(s): | 2 |
| Thread(s) per core: | 1 |
| Core(s) per socket: | 2 |
| NUMA node(s): | 1 |
| Vendor ID: | AuthenticAMD |
| CPU family: | 21 |
| Model: | 112 |
| Model name: | AMD A6-9220e RADEON R4, 5 COMPUTE CORES 2 C+3G |
| Stepping: | 0 |
| Frequency boost: | enabled |
| CPU MHz: | 1391.608 |
| CPU max MHz: | 1600.0000 |
| CPU min MHz: | 1200.0000 |
| BogoMIPS: | 3193.99 |
| Virtualization: | AMD-V |
| L1d cache: | 64 KiB |
| L1i cache: | 128 KiB |
| L2 cache: | 2 MiB |
| NUMA node0 CPU(s): | 0,1 |

### Other

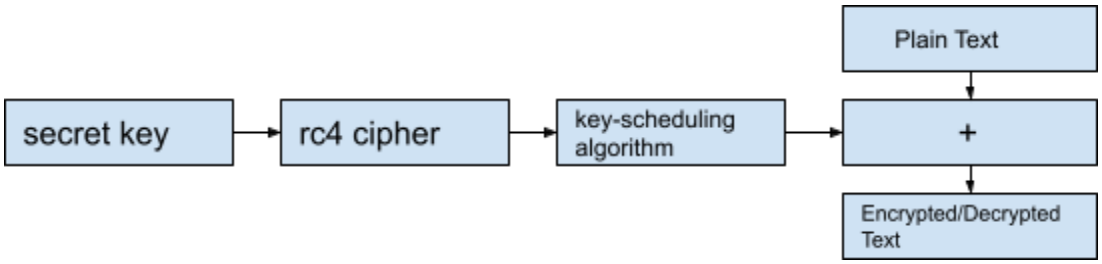
- Openssl Version: OpenSSL 1.1.1f 31 Mar 2020
- S3FS version: 1.89
- Fuse Version: 2.9.9

## Tools and Packages

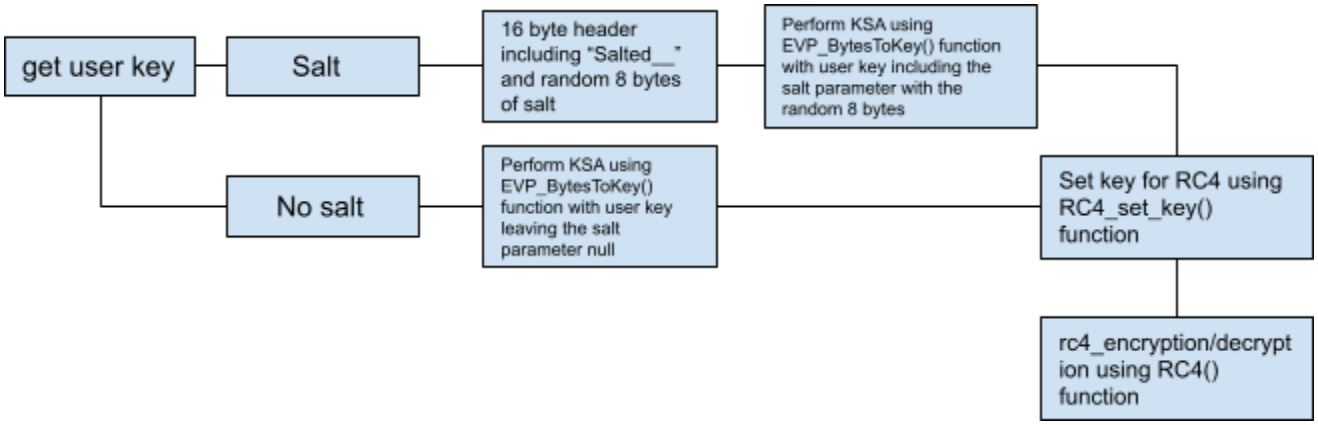
- **S3FS**: A FUSE filesystem application backed by amazon web services simple storage service (s3, http://aws.amazon.com). s3fs can operate in a command mode or a mount mode.
- **OpenSSL**: A cryptography toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) network protocols and related cryptography standards required by them. The openssl program is a command line program for using the various cryptography functions of OpenSSL's crypto library from the shell.
- **Amazon S3**: Service offered by Amazon Web Services that provides object storage through a web service interface.
- **FUSE**: A simple interface for userspace programs to export a virtual filesystem to the Linux kernel. FUSE also aims to provide a secure method for non privileged users to create and mount their own file system implementations.

## Design

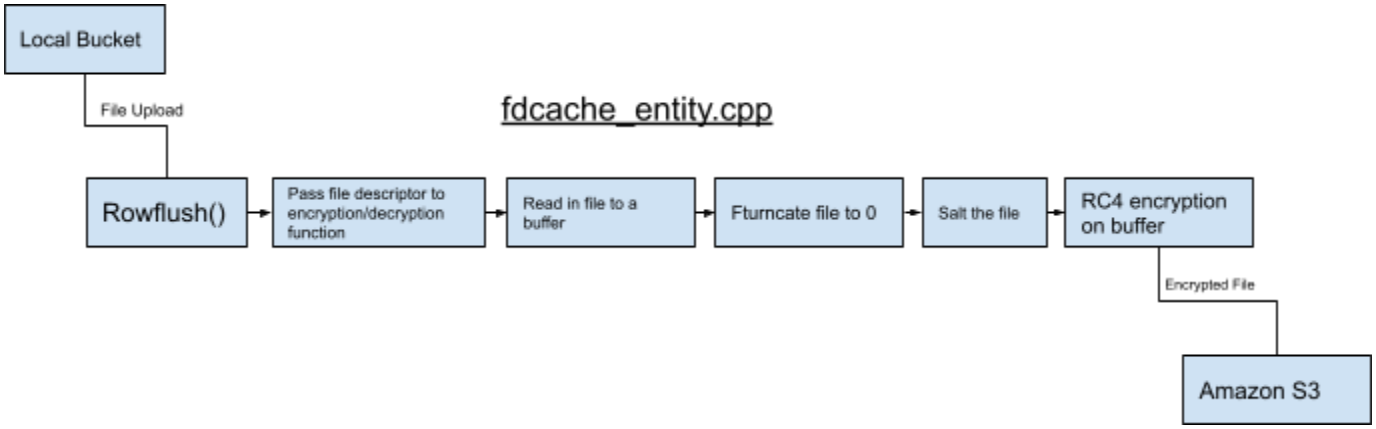
### RC4 Block Diagram



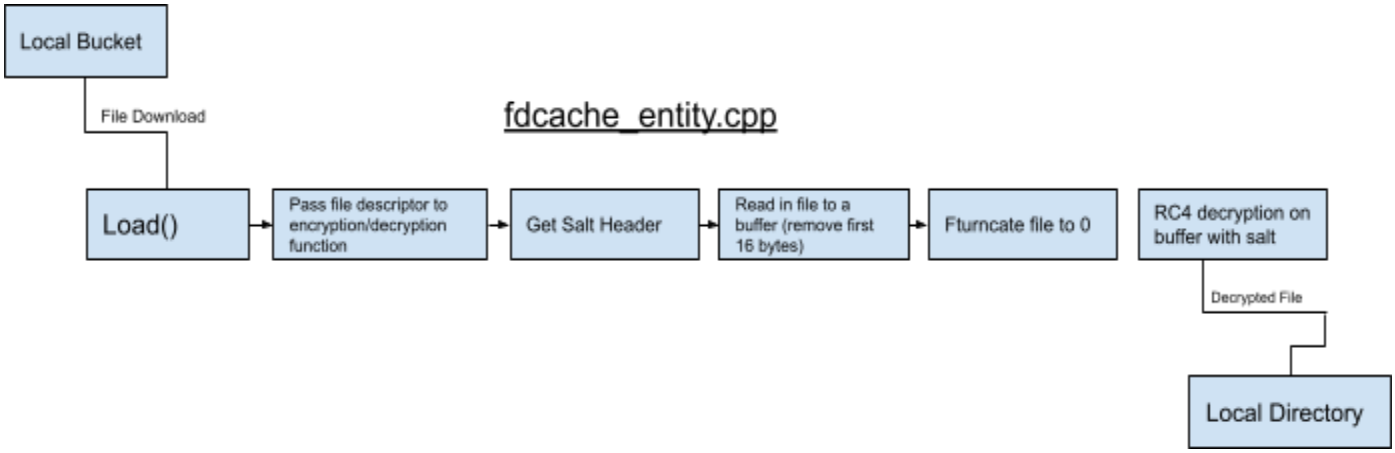

### RC4 Standalone Diagram




### S3 Encryption Diagram



### S3 Decryption Diagram




## Integration & Implementation

### Standalone RC4 Integration

In my standalone RC4, there are four parameters that are given by the user: encryption or decryption, input file name, output file name, and the key. Furthermore, the user has the option to select salt or no salt for files. However, if the parameter is not given by the user all files are salted by default. There are two buffers created, one for the input buffer and one for an output buffer. We have a buffer size of 4000000 defined so we can encrypt/decrypt files up to 4mb without any memory leaks. The input file is opened with O_RDONLY oflag as we just need to read the file contents into a buffer. The output is created, if it does not exist already, using the O_CREAT | O_WRONLY oflag with the permissions 0644 as we need to create a file that we are able to write into.

Before reaching my conditional statements, I created a bool variable named “saltrc4” that helped indicate when generating a key with EVP_BytesToKey() if the salt parameter should be included. In my first conditional statement, I handled the use input of a file being encrypted with salt. If that file was to be salted, I wrote the phrase “Salted__” to the output file with the 8 random bytes of salt too and store that in my salt variable. Thereafter, I set my bool variable to true. If a file was to be decrypted, the first thing I did was check the first 8 characters in the file to see if it contained the word “Salted__” which indicates to me the file was encrypted with salt. If that file was encrypted with salt, I would grab everything from the 8th and 16th position which is the salt value and store that in my salt variable. Since, I am decrypting with salt I would set my bool variable to true. If the file did not contain the phrase “Salted__”, I would set my bool variable to false.

Thereafter, I would generate my key to use for RC4 encryption. To generate this key as stated before, I would utilize the EVP_BytesToKey() function. The parameters in this function were: the cipher type (rc4), the hash (sha256), the salt parameter, the user key, the length of the user key, the count (1), and the computed key that is generated for RC4 use. If a file was salted and my bool variable named “saltrc4” was true, then the parameter that included salt will be included with the random 8 bytes that were generated. However, if a file was not salted and my bool variable was false, then the salt parameter would

be set to NULL. The key that was generated using the EVP_BytesToKey() function was stored in a variable. This helped with making my standalone RC4 compatible with openssl.

After the key that was generated, I used the RC4_set_key() function. This function helped set up the RC4_KEY key using the len bytes long key at data. Before starting the encryption or decryption , I needed to reset the file position to set the right read position for the input file to be read into the input buffer. To accomplish this I set up another conditional statement, if a file was salted and I was to decrypt the file I set the position of the file to be at the 16th position as I needed to read everything in the buffer without the 16 byte header

Thereafter, I read everything into a buffer. The RC4() function is then called with the RC4 key, the input buffer that contains the input file content, and the output buffer are the parameters given. The encrypted/decrypted result is stored in the output buffer. The contents of the output buffer are then written to the output file. The output file is then closed, and the file has now been encrypted.

### S3FS RC4 Integration

In the fdcache_entity.cpp, I implemented my RC4 encryption/decryption function. To implement my RC4 function into S3 there were a few things I needed to do differently from my standalone. Firstly,  since all files are going to be salted by default there was no reason to include my no salt procedure from my standalone. Secondly, since the file descriptor is being passed as a parameter in my function there was no need for my input and output file methods also. Lastly, since a user would not be passing a key every time using a terminal, there is a key that is stored by the user in a .txt file locally that is used for encryption. Decryption. I created a function get_pass() to get the contents of that .txt file and store it as the key variable to use for RC4. Also, another parameter was used in the function of int type. This parameter is used to indicate whether to encrypt(0) or decrypt(1) a file. My RC4 encryption/decryption function was called in the int FdEntity::Load() function and the int FdEntity::RowFlush() function. In the RowFlush() function I called my function e_or_d(fd, 0), since the RowFlush() function handles all uploading related calls I pass the parameter 0 to encrypt the file. In the Load() function I called my function e_or_d(fd, 1). Since the Load() function handles all downloading related calls I pass the parameter 1 to decrypt the file.

In my e_or_d() function, for encryption the first thing I did is get the file size from the file descriptor by using lseek() and use that as my size for my input and output buffer variable. I then read everything into the input buffer from the beginning of the file. Since I did  not want to add on to the existing file, I utilized the ftruncate() function. I reduced the file to 0 bytes which deleted the existing contents in the file since I did not need it anymore because it is already stored in the buffer. Since the file is salted, I need to add the first 16-byte header. To do this first, I wrote the phrase “Salted__” to the file. Thereafter, I then generated the 8 random bytes for salt and wrote that to the file. For the file to become encrypted the RC4 key needed to be generated. To complete this, I called on the EVP_BytesToKey() function passing the correct parameters with salt. This generated the key which I stored in a variable called rKey. Next, to set the RC4 key I used RC4_set_key() function. Then, rc4() which performs the actual encryption taking the rc4 key, the file size, the buffer, and the output buffer as parameters. The encrypted contents of the file are stored in the output buffer which is then written to the original file. To ensure that the first 16 bytes are not being overwritten, I use lseek() to set the cursor at the 16 byte and write the contents thereafter. Every file uploaded via the local bucket was now able to be encrypted using RC4 with salt.

Furthermore, in my e_or_d() function it can also handle decryption. It is very similar to encryption, however, there are a few extra steps taken before actual decryption. Since the file is salted when

uploading to the bucket, I need to get the salt value for the key. To get the salt value I use lseek() to get the value of everything between the 8th and 16th position and store that in a variable. Like the encryption process, I then read everything into the input buffer but instead of reading from the beginning of the file I read in everything after the 16 bytes.  Since I did  not want to add on to the existing file, I utilized the ftruncate() function. I reduced the file to 0 bytes which deleted the existing contents in the file since I did not need it anymore because it is already stored in the buffer. I then followed the same process as I did for encryption for RC4 but since the position of the file is currently at the 16th position, I used lseek() to reset the position back to 0.

## Source Code

### rc4stand.c

```
#include <openssl/rc4.h>
#include <openssl/evp.h>
#include <openssl/rand.h>

#define BUFFER_SIZE 4000000

//global variables
int argv_cnt = 1;
char *input_file_name; //hold input name
char *output_file_name; //hold output name
bool salt_status= true; //by default salt option selected
bool encrypt; //hold to encrypt/decrypt
char *eKey; // hold key
int eKeyLength; //keylength



void print_help(char **argv)
{

    printf("usage: %s -e|-d -k key -in input -out output\n", argv[0]);
    exit(1);
}

void handle_option(int argc, char **argv)
{

    ------------------------------------------------------------
    return;

}


int main(int argc, char **argv)
{
```

```
if (argc == 1 )
    print_help(argv);

while( argv_cnt < argc )
    handle_option(argc, argv);



//file read stuff
int input_file;
int output_file;
ssize_t numRead;
char input_b[BUFFER_SIZE];
char output_b[BUFFER_SIZE];
char s_check[8];


//rc4 stuff
RC4_KEY key;
const EVP_CIPHER *cipher;
const EVP_MD *dgst = NULL;
unsigned char rKey[EVP_MAX_KEY_LENGTH];
unsigned char salt[8];
char full_salt[16];
bool salt_rc4;  //checks to see the status of what encrypt of key to generate


cipher = EVP_get_cipherbyname("rc4");
dgst=EVP_get_digestbyname("sha256");
if(!cipher){fprintf(stderr, "no such cipher\n"); return 1;}
if(!dgst){fprintf(stderr, "no such digest\n"); return 1;}


//input file
input_file = open(input_file_name, O_RDONLY );
if(input_file == -1) {
    perror("Error opening input file");
    return 1;
}


//output file
output_file = open(output_file_name, O_CREAT | O_WRONLY, 0644);
if(output_file == -1) {
    perror("Error with output file");
    return 1;
```

```
    }


    //determine input file size
    int offset = lseek(input_file, 0, SEEK_END);
    lseek(input_file, 0, SEEK_SET);

    if (encrypt && salt_status)// if encrypting with salt
    {
        RAND_bytes(salt, 8); //generate 8 random bytes
        sprintf(full_salt, "Salted__%s", salt); //append 8 rand bytes generated to
Salted__ to create the full_salt string
        write(output_file, &full_salt, 16); //write full_salt to beginning of
output file
        salt_rc4=true; //salt encryption/decryption
    }
    else if (!encrypt || !encrypt && salt_status) { // if decrypt option chosen
with salt
        lseek(input_file, 8, 16);
        read(input_file, s_check, 8);
        if(strcmp(s_check, "Salted__") == 0) { //extra check to see if file
contains salt
            lseek(input_file, 8, SEEK_SET); //read/write file offset to read
saltString
            read(input_file, salt, 8); //read salt chars into salt
            salt_rc4=true; //salt encryption/decryption
        }
        else {
            salt_rc4=false;//no salt decryption
        }
    }
    else {
        salt_rc4=false; //no salt encryption
    }

    //generate key depending on if there was salt or not
    if(salt_rc4) {
        if(!EVP_BytesToKey(cipher, dgst, (const unsigned char*)salt, (const
unsigned char*)eKey, eKeyLength, 1, rKey, NULL)) //salt key
        {
            fprintf(stderr, "EVP_BytesToKey failed\n");
            return 1;
        }
    }
    else if(!salt_rc4) {
        if (!EVP_BytesToKey(cipher, dgst, NULL, (const unsigned char*)eKey,
eKeyLength, 1, rKey, NULL)) //nosalt key
```

```
        {
            fprintf(stderr, "EVP_BytesToKey failed\n"); //if BytesToKey fails.
            return 1;
        }
    }
    else {
        printf("Unable to identify if option for salt or no salt was selected! Not
able to generate RC4 Key!");
        return 1;
    }
    RC4_set_key(&key, 16, rKey); //create key


    //get position of input file
    if(salt_rc4 &&  !encrypt) {
        lseek(input_file, 16, SEEK_SET); //start input file after first 16 bytes
    }
    else {
        lseek(input_file, 0, SEEK_SET); //default inputfile from beginning of file
    }


    //write to file
    while ((numRead = read(input_file, input_b, BUFFER_SIZE)) > 0) {
        RC4(&key, numRead, input_b, output_b);
        if (write(output_file, output_b, numRead) != numRead)
            printf("write() returned error or partial write occurred");
    }

    //output file size
    offset = lseek(output_file, 0, SEEK_END);
    lseek(output_file, 0, SEEK_SET);
    //printf("*Output File size is: %d bytes\n", offset);


    if (numRead == -1){
         perror("Error with reading file");
        return 1;
    }
    if (close(input_file) == -1){
         perror("Error closing input file");
        return 1;
    }
    if (close(output_file) == -1){
         perror("Error closing output file");
        return 1;
    }
```

```
    return(0);

}
```

**fdcache_entity.cpp (encryption/decryption function)**

```
char *get_pass()
{
    int passFile;
    char *pass = (char *)malloc(100);

    passFile = open("pass.txt", O_RDONLY);
    if (passFile == -1){
        perror("PASSWORD: Error with input file");
        exit(1);
    }
    if(read(passFile, pass, 100) == -1){
       perror("Error with reading file");
       exit(1);
    }
    if (close(passFile) == -1){
      printf("error");
      exit(1);
    }
    return pass;
}


int e_or_d(int fd, int type){


    //rc4 stuff
    RC4_KEY key;
    const EVP_CIPHER *cipher;
    const EVP_MD *dgst = NULL;
    unsigned char rKey[EVP_MAX_KEY_LENGTH];
    unsigned char salt[8];


    char * pass = get_pass();
    char eKey[100];
    strcpy(eKey, pass);
    free(pass);
    int eKeyLength=sizeof(eKey-1);
```

```
    cipher = EVP_get_cipherbyname("rc4");
    dgst=EVP_get_digestbyname("sha256");
    if(!cipher) { fprintf(stderr, "no such cipher\n"); return 1; }
    if(!dgst) {fprintf(stderr, "no such digest\n"); return 1;}



    int offset = lseek(fd, 0, SEEK_END);
    lseek(fd, 0, SEEK_SET);
    printf("*Input File size is: %d bytes\n", offset);

    if (type == 0){ //encode

    unsigned char input_b[offset];
    unsigned char *output_b= (unsigned char*)malloc(offset);
    RAND_bytes(salt, 8); //generate 8 random bytes


    lseek(fd, 0, SEEK_SET); //default inputfile from begining of file
    if(read(fd, input_b, offset)  == -1 ) {
              perror("READ ERROR");
              exit(-1);
    }


    if(ftruncate(fd, 0) < 0){
      perror("FTRUNCATE ERROR");
      exit(-1);
    }

    lseek(fd, 0, SEEK_END);
    if (write(fd, "Salted__", 8) < 0){
        perror("Error opening input file");
        return 1;
    }

    lseek(fd, 8, SEEK_SET);
    if (write(fd, salt, 8) < 0){
        perror("Error opening input file");
        return 1;
    }



     if(!EVP_BytesToKey(cipher, dgst, (const unsigned char*)salt, (const unsigned
char*)eKey, eKeyLength, 1, rKey, NULL)) //salt key
     {
```

```
        fprintf(stderr, "EVP_BytesToKey failed\n");
        return 1;
 }

RC4_set_key(&key, 16, rKey); //create key
RC4(&key, offset, input_b, output_b);

//CHANGE
lseek(fd, 16, SEEK_SET);
if (write(fd, output_b, offset) < 0){
    perror("Error opening input file");
    return 1;
}

offset = lseek(fd, 0, SEEK_END);
lseek(fd, 0, SEEK_SET);
printf("*ENCODED File size is: %d bytes\n", offset);

    free(output_b);
}

else if (type == 1){ //decode
offset-=16;
unsigned char input_b[offset];
unsigned char new_buffer[offset];

lseek(fd, 8, 16);
lseek(fd, 8, SEEK_SET);
if(read(fd, salt, 8) == -1 ) {
         perror("READ ERROR");
         exit(-1);
}



lseek(fd, 16, SEEK_SET);
if(read(fd, input_b, offset) == -1 ) {
         perror("READ ERROR");
         exit(-1);
}


if(ftruncate(fd, 0) < 0){
  perror("FTRUNCATE ERROR");
  exit(-1);
}
```

```
     if(!EVP_BytesToKey(cipher, dgst, (const unsigned char*)salt, (unsigned char
*)eKey, eKeyLength, 1, rKey, NULL)) //salt key
     {
            fprintf(stderr, "EVP_BytesToKey failed\n");
            return 1;
     }
     RC4_set_key(&key, 16, rKey); //create key
     RC4(&key, offset, input_b, new_buffer);

    //CHANGE
    lseek(fd, 0, SEEK_SET);
    if (write(fd, new_buffer, offset) < 0){
        perror("Error opening input file");
        return 1;
    }

    offset = lseek(fd, 0, SEEK_END);
    lseek(fd, 0, SEEK_SET);
    printf("*DECODED File size is: %d bytes\n", offset);
    }


    return(0);
}
```

**fdcache_entity.cpp (Load Function)**

```
int FdEntity::Load(off_t start, off_t size, bool lock_already_held, bool
is_modified_flag)
{

----------------------------------------------------------------------------
            // download
            if(S3fsCurl::GetMultipartSize() <= need_load_size && !nomultipart){
                // parallel request
                result = S3fsCurl::ParallelGetObjectRequest(path.c_str(), fd,
iter->offset, need_load_size);
            }else{
                // single request
                if(0 < need_load_size){
                    S3fsCurl s3fscurl;
                    result = s3fscurl.GetObjectRequest(path.c_str(), fd,
iter->offset, need_load_size);
                }else{
```

```
                    result = 0;
                }
            }
            if(0 != result){
                break;
            }

            e_or_d(fd, 1); my implementation


--------------------------------------------------------------------------------
    return result;
}
```

**fdcache_entity.cpp (RowFlush Function)**

```
int FdEntity::RowFlush(const char* tpath, bool force_sync)
{

  ---------------------------------------------------------------------------
    if(-1 == fd){
        return -EBADF;
    }
    AutoLock auto_lock(&fdent_data_lock);

    if(!force_sync && !pagelist.IsModified()){
        // nothing to update.
        return 0;
    }

   e_or_d(fd, 0); // my implementation

 ---------------------------------------------------------------------------
        // put pading headers
        if(0 != (result = UploadPendingMeta())){
            return result;
        }
    }

    if(0 == result){
        pagelist.ClearAllModified();
    }
    return result;
}
```

## Future Improvements

Although my standalone program and my S3FS integration met all the functional requirements for this project, I believe I could make improvements to the following areas: less system calls and file size.

### Less System Calls

In the future to make improvements to this project, I would try to find ways to remove redundancies with seek system calls. I feel as if I called this system call way too much for unnecessary reasons. I believe I could have used the system calls pread() and pwrite() as they would have provided a better advantage for me. The two system calls, pread() and pwrite() ,combine both seek()+ read() and seek() + write() into one single system call. It's usually a universal truth that a program becomes a lot more efficient when it involves less system calls.

### File Size

My standalone was able to handle large files when I was testing it. However, I could not say the same for my S3FS integration. It was only able to handle files < 5mb. I understand S3 has several issues as it is very buggy, takes a lot of time downloading files and uploading files to S3, and may cost your S3 quota. However, just uploading a file less than 30mb and greater than 10 mb would feel very successful to me. I will certainly investigate how I can do this by maybe improving my strategy with the memory allocation.

## What Helped

Throughout this project, I was very lost and felt hopeless at times. However, there were several processes that I went through that helped me fight through the adversity.

- **Coming up with a timetable:** Just planning everything out in my head was not helpful at all. It sometimes just congested my head and threw me for a migraine every time I tried to plan things out in my head. Something that I found very helpful was writing down my plan on a paper. It provided me a way to check what I had planned to do and see if I was on track. Certainly, the timetable provided in the class website also helped. I just infused my personal timetable with the class timetable and followed that throughout the project. One way I infused my timetable with the class timetable was if I did not understand anything related to the class timetable, I set myself two days back and made sure I understood everything related to that topic.
- **Read and Understand:** Simply put**,** in this project a lot of reading helped me a lot. The reading done for this class and this project was very intensive. Firstly, the reading that helped me power through this project was with the openssl library. I was familiar with certain ciphers like aes256 and base64 but never understood how it worked. Reading and understanding how those ciphers worked really helped me understand RC4. If I simply did not understand a certain part, I would try to do more research on the topic by reading more on it or even watching a YouTube video. Secondly, the man command in Linux was very helpful to me. The man command displayed the user manual of any command that I could run on the terminal. For certain system calls I was not familiar with, I used the man command to understand it. Lastly, reading the source code for S3 helped me. I certainly did not know where to look to find the upload and download functions; however, after reading the documentation very thoroughly I was able to locate it.

- **Unit testing:** Throughout the coding process I encountered many issues that did not allow my code to compile or run correctly at all. Instead of running one whole program at once, I ended up separating each part into different programs and made sure those simple small programs did the right thing before incorporating it to the final product. For example, when implementing RC4 with salt into S3 I was having trouble getting the original file overwritten. I explored many paths such as writing into an output instead but that simply would not be efficient. Instead of coding in fdcache_entity.cpp for S3, I made a simple program just for encrypting files and only encrypting files. In this program, I designed a way to salt the files and overwrite the existing contents simply by using the ftruncate() function. When this was successful, I designed another program simply to decrypt files and simply decrypt files. Breaking things up into smaller tasks really helped me power through the project.

## Summary

This project was very beneficial to apply what I learned in this CSC 4420 Computer Operating Systems. To be quite honest, when I first read about the project in the beginning of the semester, I had my doubts which almost led to me dropping this class. However, I stuck with it knowing that as the semester went on, I would learn the concepts and be able to apply it. I am glad that my doubts did not force me to drop this class. This project helped me apply the many concepts I learned in the earlier classes such as memory management and file system. It was my first-time using Linux OS and I feel like I learned a great deal even though I know there is much more to learn. I became comfortable with the many system calls and using commands in the terminal rather than using the GUI in Linux. Furthermore, I learned how to use new software and programs like OpenSSL and S3FS. I always wanted to familiarize myself with AWS and I believe that I learned the basics of it through this project by using S3. Also, learning the functionality and purpose of OpenSSL and just a particular cipher was very rewarding.