*Peter Sam*

*Fv7567*

*CSC 4420 Final Project Report*

**Introduction**

For this project, we were required to use Amazon S3. Amazon S3 is a cloud based server that can be used to store files such as text documents and jpeg photos. Cloud services have become very popular in recent year. This project was a good way to be introduced to the code behind a cloud. Amazon Web Service (AWS) has a product called Amazon S3. With Amazon S3, a user can set up a bucket with user defined policies. The user can create as many buckets as he/she wants. Inside each bucket, a user can upload a file from his desktop, or he can download a file into his desktop. Amazon S3 can be very useful to save information, and it is useful to get more storage for any documents or photos.

In the modern age, cloud storage has become something that is not only convenient, but it has become something that is really essential. A lot of people use cloud storage nowadays. Major businesses and individual people all use some sort of cloud. It is very popular. Cloud storage has become so important because people store so much information on their computer and the internet. Cloud services is very useful because it provides more storage to people who don’t have enough storage on their computer. Cloud services also allow people to share their information across multiple platforms like a smartphone or a laptop. In this project we used the Amazon S3 cloud. To use Amazon S3, it is important to have internet connection. In order to access the cloud, the use has to enter the following site: <https://aws.amazon.com/s3/>. After creating an account, the user can create a bucket to save any personal information.

One of the problems with cloud services is that it is not as secure as saving pictures or document’s locally on a desktop. When a user sends information to a cloud, they are pretty much sending their information to the internet. Everyone can have access to the internet, so it is very easy for people to hack personal information from a cloud. So although cloud services are very useful, they can also be very susceptible to the danger of being hacker and losing a lot of personal information.

We used Amazon S3 in this project. We had to download a s3fs-fuse use several commands on github, and then s3fs had to be mounted on our computer. The main goals in this project was to encrypt and decrypt files in s3fs. When I first started this project I was a little bit confused. I felt so lost in the first two weeks. I never had a project before where I had analyze the source code and modify existing source code. Usually for my previous computer science classes at Wayne State University I had to write code all from scratch. For this project, I actually sit down and spend hours trying to make sense out of the code.

**Project Goals**

The goal for this project was to understand the functionalities of the Amazon S3 cloud and to get familiar with s3fs-fuse code. In this project, the professor asked us to design and implement a stackable file system (Amazon S3) running on Linux. This project focused a lot on how to encrypt and decrypt files using RC4. We had two main tasks in this project. The first main task was to create a standalone utility. The standalone utility allows the user to encrypt and decrypt files locally on a desktop by writing a command in the terminal similar to the OpenSSL commands. The second main task of this project was to integrate our RC4 code in s3fs-fuse code so that we can send encrypted data to the Amazon S3 cloud.

To properly use the integrated s3fs-fuse code, it is important to install and mount s3fs-fuse in a computer. Once RC4 is integrated and s3fs in mounted, a user can copy a file into his s3fs mount. Then the user can sign in to his Amazon s3 account and access his bucket to look for the file he copied to his mount point. Once the user tries to open his file in the website, the file will be encrypted, and there will only be nonsense presented on the screen. To decrypt the file, the user has to download the file from the website. Once downloaded and transferred to the mountpoint. The file should be properly be decrypted.

For this project we were given a project timeline. The project timeline was very useful, and I referred to it a lot. It give us the tasks to do each week. The project spanned a total of 6 weeks. The first 4/5 weeks consisted of writing the code, and the last two weeks were focused on demoing the project, finishing up code/debugging, and completing the final report for the project. For the most part, I felt like I was on track for the project. I was confused the first two weeks, but after that I was able to get a lot more work done.

1. **Mar. 19 week**: understand s3fs source code, find out where the upload and download components are
2. **Mar. 26 week**: learn how to change data content in s3fs with a simple function, e.g., exclusive or 0x0011 to each byte before upload and then after download, the resulting file should be the same as the original one before upload
3. **Apr. 2 week**: understand RC4 part source code in OpenSSL, and test with rc4 commands in OpenSSL
4. **Apr. 9 week**: Implement a simple stand-alone utility to encrypt and decrypt a file using RC4; start integrating RC4 into s3fs, store encryption key locally
5. **Apr. 16 week**: Further integrate RC4 into s3fs; various tests; preparing presentation
6. **Apr. 23 week**: finalize secure-s3fs 0.01; report writing
7. **Apr. 29**: Report due

**System Information**

For this project. I had to use my own personal computer to write and test the code for the project. I also had to download certain software. The following are the description of all the specs of my computer and a detailed list of all the software I used:

**Computer:** 2015 Macbook pro

-CPU: 2.9 GHz intel core i5 processor

-Memory: 8GB 1867 MHz DDR3

-Storage: 500 GB SSD

-MacOs

**Software:**

### Virtual Box 5.2.10:

### Virtual box is a free virtual machine. It is a x86 and AMD64/Intel64 virtualization product available to the public for free. I used this software to run Linux on my computer. I reserved 20 GB of storage for my virtual machine.

### Ubuntu 16.04.4 (64-bit)

### This is the Linux distribution I installed it in virtual box.

### S3FS-FUSE

### Amazon S3 is Amazon’s cloud service program. The s3fs package is the code for Amazon Web Service’s cloud. It is a FUSE based file system. The encryption/decryption modifications can be made in the files of the source code from s3fs-fuse package.

### OpenSSL 1.0.2

### -OpenSSL is a software used for secure communications over computer networks. OpenSSL has many cryptography. For this project, it is necessary to implement RC4. OpenSSL has the RC4 library which can be implemented in s3fs-fuse. It is already pre-installed in Ubuntu

### FUSE 2.9.5

### -FUSE (Filesystem in Userspace) is an interface for userspace programs to export a filesystem to a Linux kernel. This is necessary when creating a mount point.

### Tools and Packages

### For this project, I had to use several tools and packages. As mentioned in the previous sections, I have downloaded and installed several software program such as s3fs-fuse, OpenSSL 1.0.2, and FUSE 2.9.5. All the necessary source code I needed were in the s3fs-fuse and the OpenSSL libraries. The tools needed for RC4 and MD5 were found in the OpenSSL package. For the RC4 integration part, it was very important to use the source code in the s3fs-fuse package. I used gedit as the text editor to write my C++ code for the standalone.

### Design

### The first thing I did when I started the project was sign for an AWS account. I then created my bucket, and I established the policies for the bucket. After that I had to download and install s3fs-fuse. I used the following website: <https://github.com/s3fs-fuse/s3fs-fuse>. I used the github URL to know what commands to type in the terminal in order to successfully install s3fs.

I successfully installed and mounted s3fs on Ubuntu. It was difficult to understand how to successfully mount it to a directory, but I realized that I was supposed to mount s3fs every time I restart my virtual box VM. I was able to successfully exchange files between the local mounted bucket and the online bucket.

### The next part of the project was to understand the s3fs code. This proved to be one of the most time consuming parts of the project because there were many source code files. In many of the source code files, there were over thousands of lines of code. I was instructed to find where the read and write functions were in the code. Eventually after many attempts using “control + f” to find the read write functions. I was able to find the read and write functions in the s3fs.h and the fdcache.h files. This took me to the fdcache.cpp file in s3fs-fuse. I talked to my classmates and they all confirmed that the read and write functions were in fdcache.cpp. The two following functions I manipulated in the fdcache.cpp are:

* Ssize\_t FdEntity:: Read(char\* bytes, off\_t start, size\_t size, bool force\_load)
* Ssize\_t FdEntity::Write (const char\* bytes, off\_t Start, size\_t size)

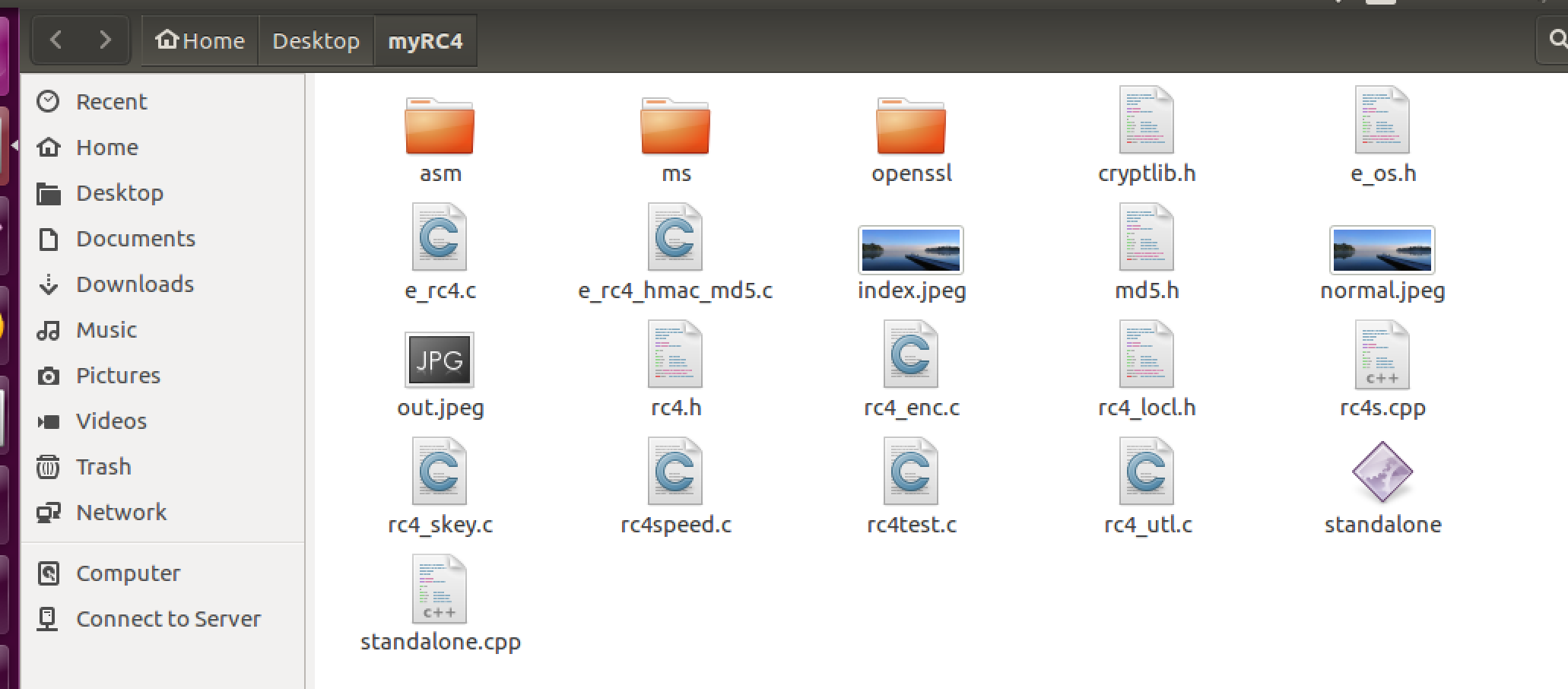
Encryption was able to happen in the read function and decryption was able to happen write function. The rowFlush function and the load function were also able to run encryption and decryption, but I have found that encryption and decryption works better in the read and write functions. I decided to use the read and write functions because it made more sense to me, and I found it easier to use.

The process of how files were being read and written to in Amazon S3 made more sense to me when I properly read the read and write functions in the fdcache.cpp. I learned that the program takes the file descriptors of the files that are being sent to the cloud. The program will take in a file and read. As soon as the file is read, it will be placed in a char\* array, and it will then get the size of the file. After some error handling, it finally reads and writes to file. To read and write, it uses the pread() and pwrite() functions in the code.

After looking through the read and write functions in the fdcache.cpp file. I downloaded OpenSSL and all the necessary source files found in OpenSSL using the Linux terminal. After looking through the RC4 library, I was able to get a better understanding of how RC4 works. To run encryption and decryption in OpenSSL, it is necessary to run the following command line in the terminal:

-openssl -rc4 -e/-d -in (name of file) -out (name of file)

After trying to get a better understanding of the OpenSSL source package. I tried to figure out ways I could integrate in my s3fs mountpoint. Another thing I could focus on after reviewing the OpenSSL code and the OpenSSL commands was my standalone code. The purpose of the standalone could is that it must be built against the OpenSSl commands. I first created a folder called myRC4 in my desktop to contain my standalone and all the necessary folders and files that were necessary to run my standalone code. I created my standalone code in gedit. I wrote the code in C++. Before I could make any actual modifications to the code. I had to include all the necessary libraries. After that I had to copy certain files and folder in my myRC4 folder. A picture of all the files and folders I included can be found in the picture below:



After including all the necessary libraries, files, and folders, I was able to actually start writing my code. I had to copy this files and folders from OpenSSL. This was one of the most time-consuming parts of the project. The standalone code is the bulk of the project, and it was essential to have standalone code working smoothly. I had to meet up with my TA multiple times in order to get my standalone code working properly. My standalone code was designed in a certain way so that it could encrypt and decrypt and files given in the terminal. There were seven main steps in my implementation of the standalone code. The following seven steps are:

1. Accept three arguments from the terminal (./standalone inputFile outputFile). This command can be used to either encrypt or decrypt a file. The encrypted/decrypted file can be found in the myRC4 folder after the command has executed in the terminal. The standalone can work with either .JPEG files or even .txt files works as well.
2. Open the files
3. Read the files
4. Create the encryption function to use RC4 to encrypt the files that were open and read.
5. Check for errors.
6. Close the files
7. Free memory.

Before I could execute my code, I had to create an executable for my standalone.cpp file. I had trouble figuring out what was the proper command to create the standalone executable because whenever I tried to create an executable, the terminal would give me several errors. The proper command to create an executable for the standalone is the following command:

g++ standalone.cpp -o standalone -lssl -lcrypto

The above command got me to create an executable on openssl After the code was executed I could successfully decrypt and encrypt files using my code. If I wanted to encrypt or decrypt a file, I just have to type in 3 arguments. The program can only take three arguments otherwise there will be an error. I put some error handling in the code so that would be prevented. The following shows an example of what type in the terminal when using my executable for the standalone code in order to encrypt/decrypt a file:

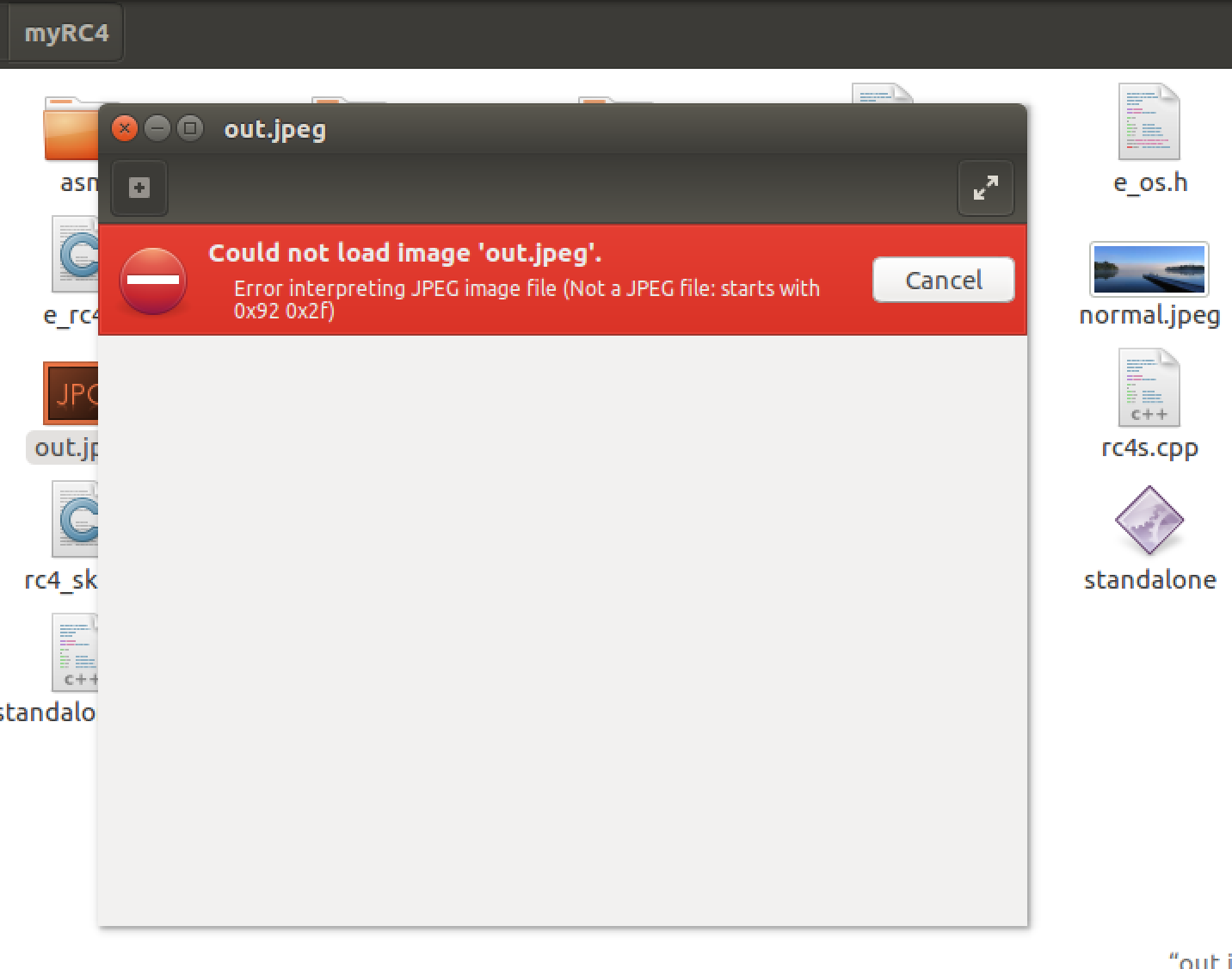
To encrypt:

./standalone inputFileName outputFileName

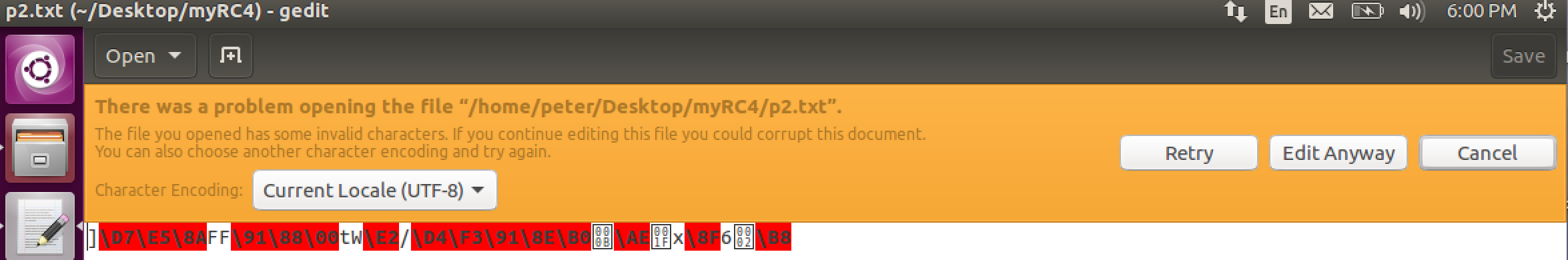
To decrypt:

./standalone encryptedFileName decryptedFileName

After either one of those commands are run into the terminal, the code will produce an output file inside the myRC4 folder. If a file is encrypted file, there will be an encrypted file in the myRC4 folder. The file will be unreadable by the computer. An example of an encrypted JPEG image can be seen below.



The file cannot be opened because it is encrypted. This is how a .JPEG file looks like after it is encrypted. The name of this .JPEG image is called out.jpeg because that is what was put in the terminal when the executable was activated. A text file looks a little different because it does not use the image viewer application to view the encrypted file. Instead the encrypted text can be seen in the gedit text editor. An example of an encrypted text image can be seen in the picture below:



### Before I can use my standalone code, I have to first type in the following command in the terminal:

### -cd Desktop/myRC4

### This takes the terminal to the myRC4 folder with the necessary code to run encryption/decryption. Any file (.JPEG or .txt) that has to be encrypted/decrypted must be left in the myRC4 folder before it can be modified by the standalone code. Any file that is placed outside this folder will not be able to be encrypted/decrypted unless the path is specified in the terminal with the standalone command.

### I was able to demo my code in front of the class, and it was able to perform well. When I first demoed my project to the class I was able to get my standalone running, and I was able to do the integration of my standalone with s3fs properly. The errors I had was that my standalone decrypt and thus my standalone integration with s3fs could not decrypt either. Another problem was that my standalone was not compatible with OpenSSL. After my demo, I was able to correct most of those errors. I was able to get my standalone decrypting after a few hours of debugging. I then was able to correct my integration with s3fs. The part of the project that I had difficulty with was with OpenSSL. My standalone had trouble with being fully compatible with OpenSSL.

### Integration

### After my standalone.cpp was completed, I was finally able to finish my integration with s3fs-fuse. Once the standalone is done, most of the work for this project done. Only parts of the standalone code has to be copied to the fdcache.cpp file. I knew I had to modify the fdcache.cpp. In the beginning, I was able to get some part of the integration done, but after I created a standalone utility, I was able to integrate with s3fs. At first, I tried to work with read and write functions in the fdcache.cpp because that was where all the download and upload components. When I first started the integration part of the project, I tried hard coding the RC4() and the RC4\_set\_key() functions in the read and write functions of fdcache.cpp, but that was giving me some problems. Once I implemented my standalone into the fdcache.cpp, It worked much more smoothly.

### When I first implemented my standalone into my s3fs-fuse, I had to first copy standalone folder (myRC4) into the src folder of the s3fs-fuse package. I had to copy it into the src folder of s3fs-fuse because that was where the fdcache.cpp files was. Then I had to include some more libraries inside the fdcache.cpp file in order for the program to recognize my standalone utility when it runs the code. I then had to copy the encryption function of my standalone into the beginning of the fdcache.cpp file. Now that my encryption function is fully integrated in to the fdcache.cpp file, I am able to call my function in download and upload function of the fdcache.cpp file ( the read and write functions). First, I tried calling my encryption/decryption function inside the read and write functions of fdcache.cpp, but I was running into trouble. It wasn’t encrypting or decrypting. After doing some more research, I realized I was able to call my standalone’s encryption/decryption function in the functions below:

### Int FdEntity::RowFlush(const char\* tpath, bool force\_sync);

### Int FdEntity::Load(off\_t start, size\_t size);

### After I called my encryption function in those two functions I was able to properly encrypt and decrypt files. I had to call my function in certain parts of the Load and RowFlush functions of the fdcache.cpp code. The way my integration worked was that every time a file was sent to my mountpoint, it would be encrypted. So when I go to the Amazon S3 website, I am not able to open file because it is encrypted. When I download the file from the website and put it back into my mountpoint, the encrypted file becomes decrypted. Both .JPEG images and text files can be encrypted/decrypted with intergrated s3fs-fuse code I implanted. A picture of an encrypted .JPEG image in the Amazon S3 website can be found in the image below.

### Implementation Details

### The following is the highlights of the standalone.cpp code I implemented. I had to include several libraries from OpenSSL because they were necessary to run my standalone. In my main function, I had two parameters in my main function (int argc, char\*argv[]). Those two commands were useful for taking in arguments from the terminal. From those arguments, I could open and read files. I also had hard coded my own key stored in a string called data. This key is the key that will be used for keystream in RC4 in order to encrypt and decrypt the file.

#include <iostream>  
#include <fstream>  
#include <string>  
#include <cstdlib>  
#include <stdlib.h>  
#include <stdio.h>  
#include <fcntl.h>  
#include <unistd.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
//#include <openssl/rc4.h>  
#include <openssl/crypto.h>  
#include <openssl/evp.h>  
#include <openssl/rand.h>  
#include "cryptlib.h"  
  
//#include <cryptlib.h>  
#include "md5.h"  
#include "rc4.h"  
#include"rc4\_locl.h"  
#include "rc4\_enc.c"  
#include "rc4\_skey.c"  
  
using namespace std;  
  
void encrypt(int fd,int fd2);  
//unsigned char \*MD5(const unsigned char \*d, size\_t n, unsigned char \*md);  
//void decrypt();  
int main(int argc, char\* argv[] )  
{  
    //string data = "encryptkey";  
    int fd1;  
    int fd2;  
    if( argc == 3)// accepts 3 arguments  
    {  
        fd1 = open(argv[1],O\_RDWR,0700);//open file (fd1)  
        fd2 = open(argv[2], O\_CREAT | O\_RDWR,0700);//open output file (fd2)  
        //put error message  
        encrypt(fd1, fd2);//encryption  
          
    }  
    //error handling  
    else if(argc != 3)  
    {  
        perror("Error: not enough arguments");  
    }  
    /\*else if( argc == 6 && arg[2] == "-d")  
    {  
        void decrypt();  
    }  
    \*/  
    if((fd1 = open(argv[1],O\_RDONLY)) == -1 || ((fd2 = open(argv[2], O\_CREAT | O\_RDWR,0700)) == -1))  
    {  
        perror("Error: incorrect use of arguments");  
    }  
    close(fd1);//close files  
    close(fd2);  
    return 0;  
}  
void encrypt (int fd, int fd2)// another char \*bytes?  
{;  
    string rc4key = "potroast";// key to encrypt  
    int length = rc4key.length();// length of key  
    int size;  
    //const int fdSize = 4096;  
    //char buffer[fdSize];  
    //reading from file  
    size = lseek (fd,0, SEEK\_END);// size of input files  
    unsigned char\* infile = (unsigned char\*) malloc(size+1);// char\* to hold input file info  
    memset(infile,0,size+1);// allocate memory  
    lseek(fd,0, SEEK\_SET);  
    ssize\_t count;  
    //read(fd,infile, size);  
    count = read(fd,infile, size);  
      
    unsigned char\* md5key =(unsigned char\*) malloc(length);// creates a md5 key to work with openSSL  
    MD5((const unsigned char \*) rc4key.c\_str(),length, md5key);// md5 function  
    RC4\_KEY key;  
    unsigned char\* outfile = (unsigned char\*) malloc(size+1);  
    memset(outfile,0,size+1);  
    RC4\_set\_key(&key, length,md5key);  
    RC4(&key, size,infile, outfile);  
    lseek(fd2, 0, SEEK\_SET);  
    write(fd2, outfile,size);   
    free(outfile);// free memory  
    free(infile);  
    free(md5key);  
      
      
}

### The code below is the code I implemented in the fdcache.cpp. The following code is used for the integration of s3fs with my standalone. The code I put into the fdcache.cpp file is similar to the code I created in the standalone.cpp. The main difference is that I did not include the main function from the standalone into the fdcache.cpp file. I only included the encryption function, and I called the encryption function in Int FdEntity::RowFlush(const char\* tpath, bool force\_sync) and Int FdEntity::Load(off\_t start, size\_t size). I called the encryption function by taking the file descriptor as a parameter in the function. So I placed encrypt(fd) in the Load and RowFlush functions in the appropriate places.

#include <openssl/rc4.h>  
#include <openssl/crypto.h>  
#include <fstream>  
  
#include "common.h"  
#include "fdcache.h"  
#include "s3fs.h"  
#include "s3fs\_util.h"  
#include "string\_util.h"  
#include "curl.h"  
#include "myRC4/md5.h"  
void encrypt (int fd)// another char \*bytes?  
{  
    //string pass;  
    //cout<< " enter password: " <<endl;  
    //cin >> pass;  
    string rc4key = "potroast";// key to type in openssl command  
    int length = rc4key.length();  
    int size;  
    //const int fdSize = 4096;  
    //char buffer[fdSize];  
    //reading from file  
    size = lseek (fd,0, SEEK\_END);// size of file  
    unsigned char\* infile = (unsigned char\*) malloc(size+1);// char\* for file being encrypted/decrypted  
    memset(infile,0,size+1);//allocate memory  
    lseek(fd,0, SEEK\_SET);  
    //ssize\_t count;  
    //read(fd,infile, size);  
    read(fd,infile, size);// read input file  
      
    unsigned char\* md5key =(unsigned char\*) malloc(length);// md5 to work with openssl  
    MD5((const unsigned char \*) rc4key.c\_str(),length, md5key);  
    RC4\_KEY key;  
    unsigned char\* outfile = (unsigned char\*) malloc(size+1);// char\* for output file  
    memset(outfile,0,size+1);  
    RC4\_set\_key(&key, length,md5key);  
    RC4(&key, size,infile, outfile);  
    lseek(fd, 0, SEEK\_SET);  
    write(fd, outfile,size); // write to file  
    free(outfile);// free memory  
    free(infile);  
    free(md5key);  
      
      
}

**Future Improvement**

### I was able to get the bulk of this project done, but there were still things that was not able to finish in this project, and there are some things that I could have made better in this project. I was able to get the integration part of the code down for the most part. The only problem I had in the integration part of the code was that I was getting warnings whenever I compiled the s3fs-fuse code. It would still let my mount my bucket, and I still able to get files encrypted/decrypted from Amazon S3. I don’t think that fdcache.cpp liked that fact that I used read() and write() functions. I don’t even know why I got warnings because of it. I think I should have used the pread() and pwrite() functions in my implementation when I was reading and writing to files. I chose read()/write() because they made more sense to me, and it worked fine in my standalone code. I tried implementing pread() and prwrite(), but I was having trouble, and I didn’t have enough time. If I had more time to work on this project, I could’ve wrote a better integration.

### Another problem I faced in this project was with my standalone. My standalone was not compatible with OpenSSL for some reason. Whenever I encrypted a file, I could not decrypt it using OpenSSL. I tried to implement the md5() function in the standalone.cpp to remediate this problem, but it wasn’t working for me. This was another problem because of time. If I had enough time, I might have been able to get my standalone utility compatible with OpenSSl.

### Another problem I faced was with the diff command. According to the project guidelines, whenever I decrypted a file it is supposed to be exactly the same as the original. Although my decrypted files looked exactly the same as my original file, the diff command kept on saying that the binary files differ. So they weren’t the same. I don’t exactly what exactly what wrong with that, but they still looked very similar. It may have been because I did not encrypt using salt. My standalone utility used no salt for the encryption. I couldn’t figure out how to encrypt with salt. That also could be a reason why my standalone utility was not compatible with OpenSSL.

### One of things I could’ve improved for this project was that I should have included more error handling. Error handling was the last thing I implemented in my code, and feel like I should have done more error handling because that is very good programming etiquette.

### Summary

### This was definitely one of the biggest and most in-depth computer science project I have ever done. At first this project was very overwhelming. This was a very time-consuming project. I was so lost on this project for the first two weeks, and I definitely learned some valuable lessons along the way. One of the major lessons I learned after completing this project was time management. I felt that if I spent more time on my code that I would have been able to accomplish all the tasks for this project. I should have been more confident when I started this project in order to get as much done as possible. This project also taught to value office hours. I went t.a.’s office hours almost every week once I started this project because I had so many questions. I was able to receive a lot help from t.a., and it helped me a lot for the project.

### This was a very interesting project, and I definitely learned a lot. This project is a very good project to put on my resume. This was the first time I had to sit down and analyze code from a source package and then make my own implementation. Even though I found this project really annoying in the beginning, I liked this project because it had a lot of real world application.I feel proud of the work I’ve done in this project. This is definitely a good skill to have in the work force.