Integration of RC4 to S3FS and Standalone RC4 Utility File

Final Report

Written by Olubukola Akintoroye

April 29th, 2018

CSC 4420

Content

1. Introduction
2. Project Goals
3. System Information
4. Tools and Packages
5. Integration
6. Implementation Details
7. Future Improvement
8. Summary

Introduction

This report covers the following topics:

* Project Goals: Discuss the overall project and explain the end result of the project.
* System Information: Displays the system information of technology and programs that were used during this project.
* Tools and Packages: Gives a brief overview of the tools and packages that have been used during this project.
* Integration: Covers my experience while doing this project, this includes any accomplishments I have made or goals I have failed to meet. I have also included screenshots of my results.
* Implementation Detail: Displays all the code that has been written and a description of how each function works.
* Future Improvement: Covers improvements that I could not work on during the allocated time frame.
* Summary: Covers my overall experience with the project and what I have learned.

Project Goals

There are two major goals in this project. The first goal is to fully integrate RC4 into S3FS using the cloud storage service Amazon S3. After full integration, files should be able to upload to the local directory unencrypted and encrypt once in the Amazon S3 bucket. 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.

The second goal is to create an independent standalone of RC4. The standalone of RC4 should be able to encrypt and decrypt a file using a selected password. The standalone should also work with OpenSSL, i.e. if the standalone encrypts a file, then OpenSSL should be able to decrypt the encrypted file and produce the same results as the original file. An additional goal is to make the standalone work with the -salt and -nosalt flags.

System Information

System information for Windows Computer:

* Edition: Windows 10 Home
* Version: 1709
* OS Build: 16299.371
* Processor: Intel® Core™ i5-5200U CPU @ 2.20 GHz
* Installed RAM: 8.00 GB
* System type: 64-bit operating system, x64-based processor

System information for Virtual Box:

* Edition: ubuntu 16.04 LTS
* Memory: 1.9 GiB
* Processor: Intel® Core™ i5-5200U CPU @ 2.20GHz
* Graphics: Gallium 0.4 on llvmpipe (LLVM 4.0, 256 bits)
* OS type: 64-bit
* Disk 9.4 GB

System information for other programs:

* S3FS version: V1.83
* OpenSSL version: 1.0.2g

Tools and Packages

The following is a list of tools and packages used during this project:

* S3FS: A FUSE filesystem application backed by Amazon web services simple storage service. (Description from: https://linux.die.net/man/1/s3fs)
* OpenSSL: A robust, commercial grade, and full featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It is also a general-purpose cryptography library. (Description from: https://www.openssl.org/)
* Amazon S3: provides storage through web services interface. It is an object storage built to store and retrieve any amount of data from anywhere. (Description from: https://aws.amazon.com/s3/)
* Virtual Box: A powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. (Description from: <https://www.virtualbox.org/>)
* Visual Studio Code: A source code editor developed by Microsoft for Windows, Linux and macOS. (Description from: https://en.wikipedia.org/wiki/Visual\_Studio\_Code).

Integration

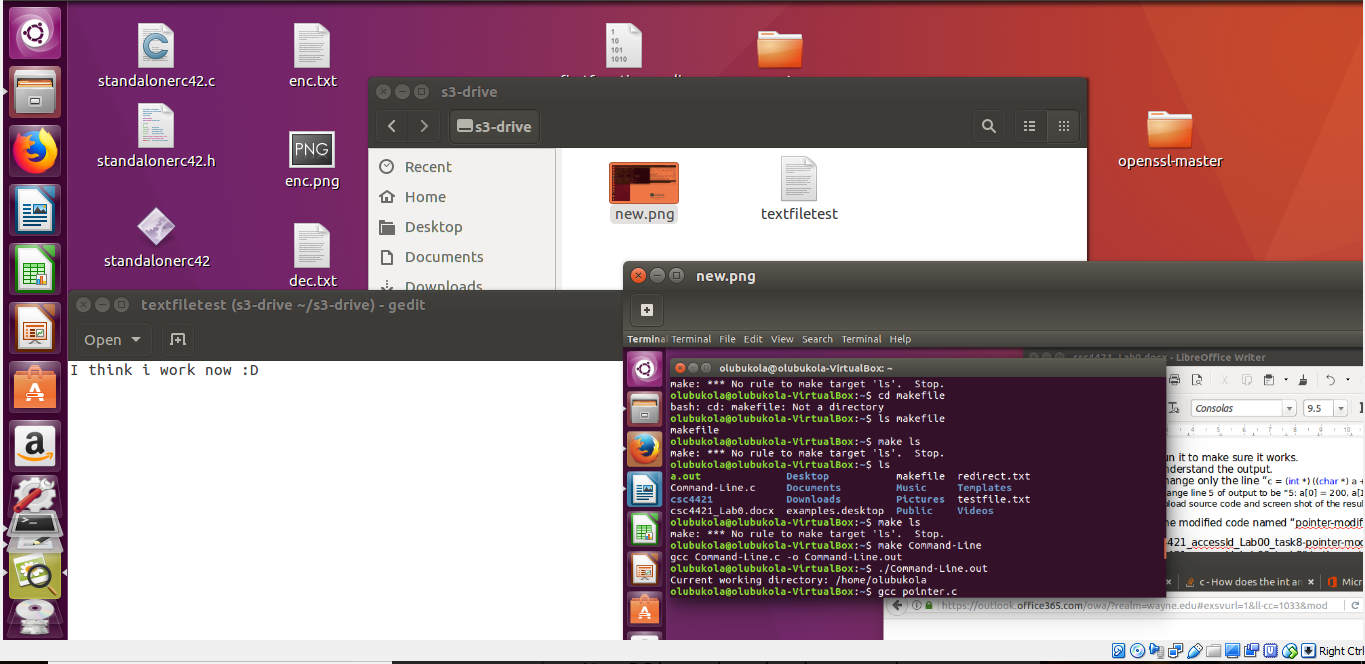
During the duration of this project, I successfully integrated RC4 to S3FS. S3FS can now take a file of any extension type and store the encrypted or decrypted version of it in the Amazon S3 bucket. While integrating RC4 to S3FS, I ran into a few problems. 

Figure 1: Original .txt and new.png being placed into local directory bucket s3-drive

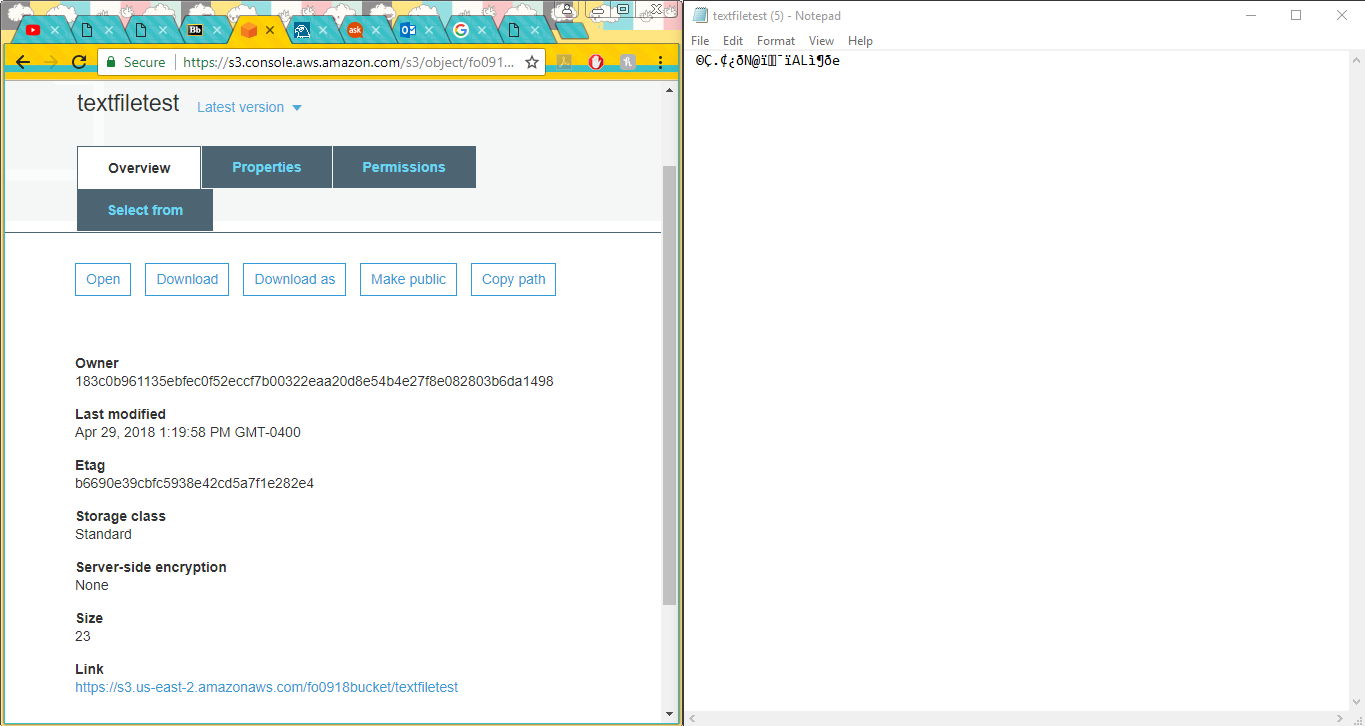


Figure 2: textfiletest is encrypted in the Amazon S3 bucket

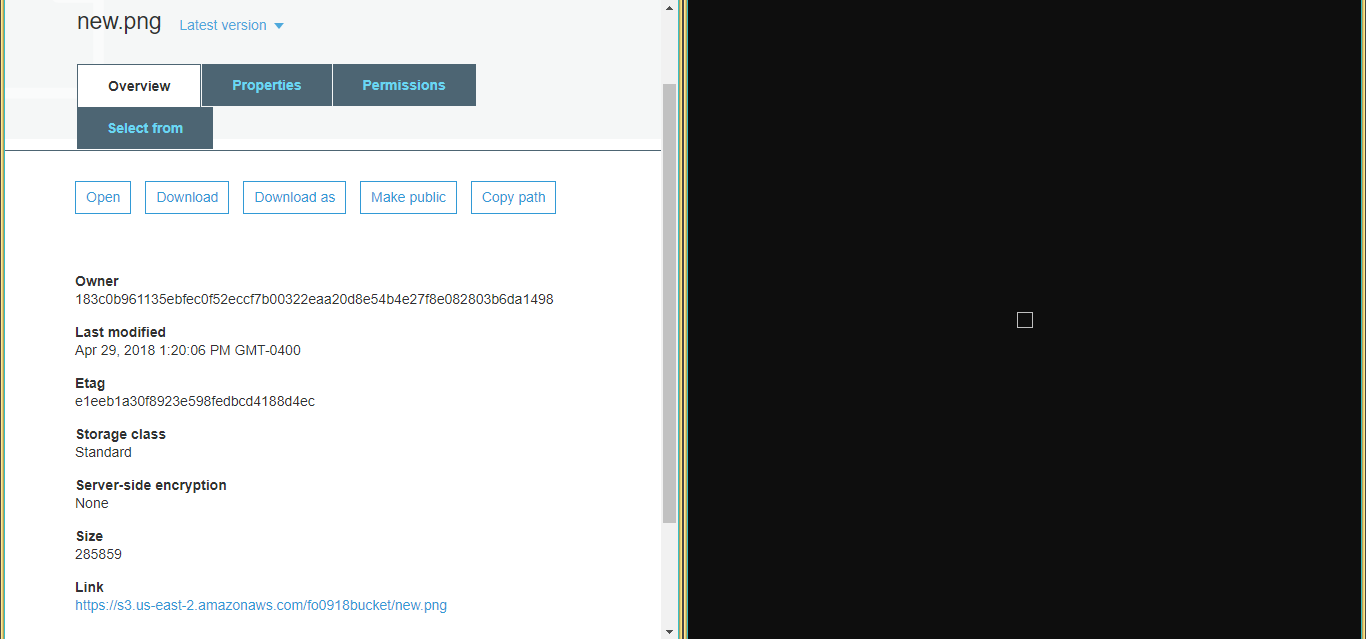


Figure 3: new.png is encrypted in the Amazon S3 bucket

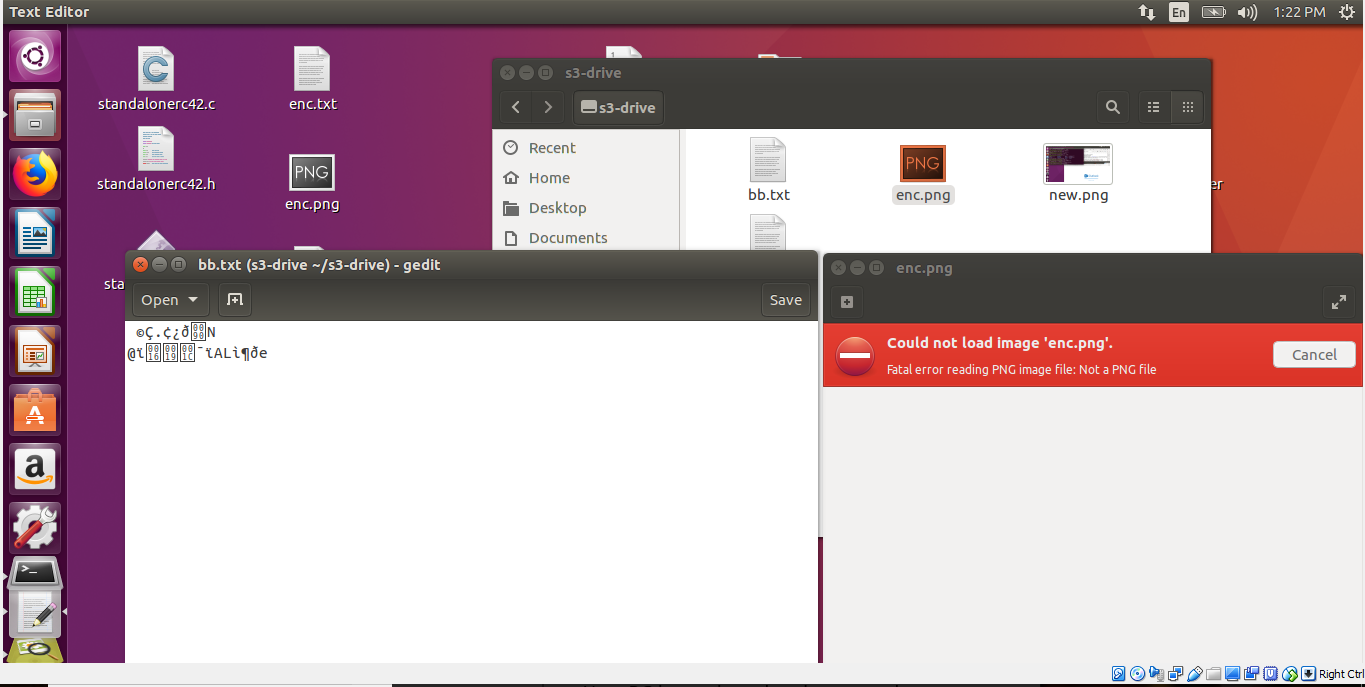


Figure 4: Encrypted files bb.txt and enc.png placed in local directory bucket s3-drive

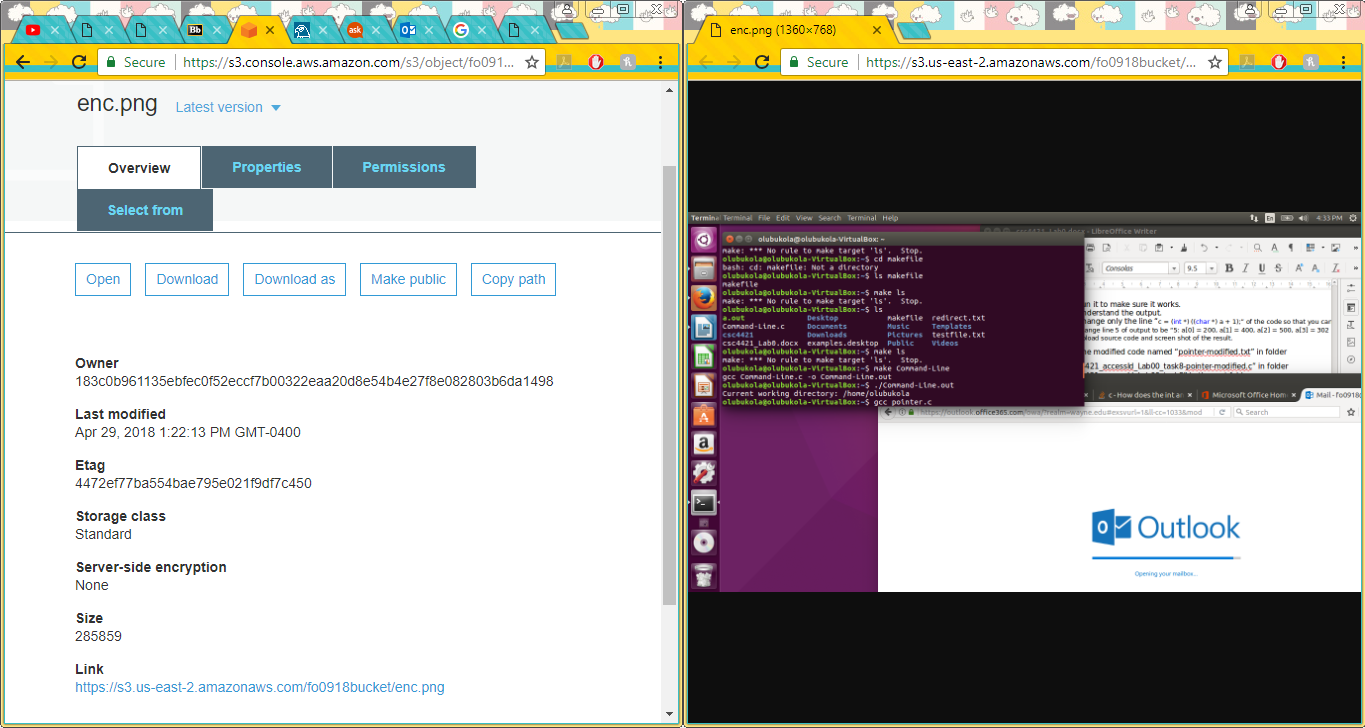


Figure 5: enc.png decrypted in Amazon S3 bucket

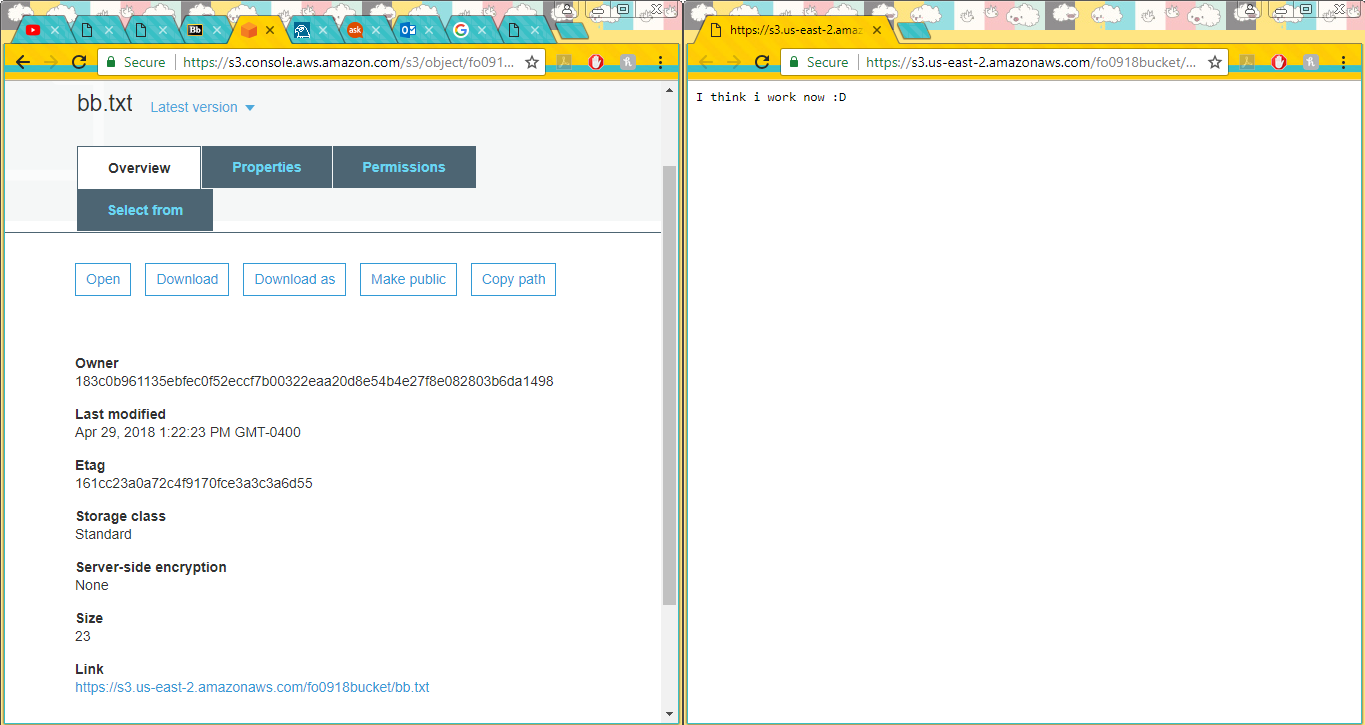


Figure 6: bb.txt decrypted in Amazon S3 bucket

The first problem was correctly mounting the mount point to my local directory. The error produced from this was “Unhandled error message: Error when getting information for file ..: Transport endpoint is not connected”. This error was solved by reinstalling s3fs and retrying the mounting process once again.

The second problem was figuring out how to compile source code within S3FS. Compiling the S3FS source files was not clear to me until the final days of the project. I attempted to use command lines such as: g++ -I/usr/include/libxml2/ s3fs.cpp s3fs -D\_FILE\_OFFSET\_BITS=64 and adding ‘-d’ and ‘-f’ to the command line for mounting the mount point. Both were incorrect, unfortunately. It was later explained to me that the following commands compiled all S3FS files:

cd s3fs-fuse

./autogen.sh

./configure

make

sudo make install

The third problem was displaying encryption in the Amazon S3 bucket. I confirmed that my code for encryption/decryption was correct but the results did not display properly. I found that when using lseek to determine the number of bytes in a file (SEEK\_END) that lseek needed to be reset to the first byte using SEEK\_SET. If SEEK\_SET was not used after reading the file, it would add the encrypted content to the bottom of the original file instead of overwriting it.

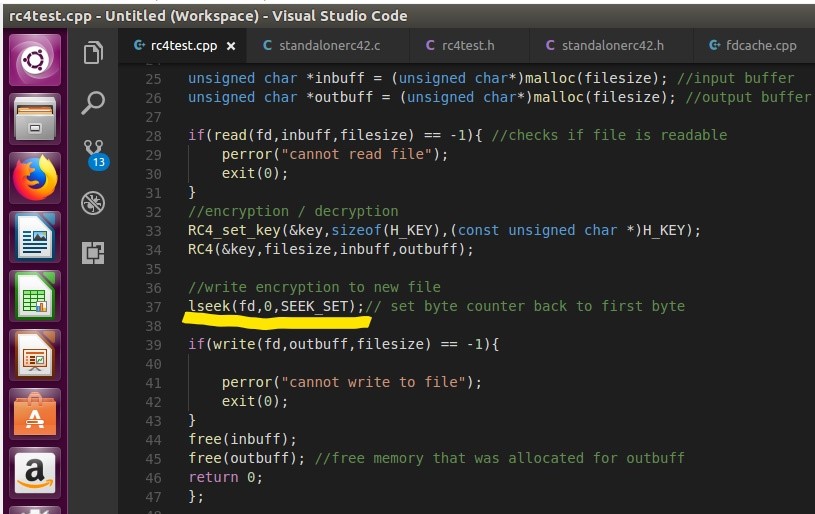


Figure 7: Insertion of lseek

The standalone RC4 utility file was not successfully completed. The standalone by itself can encrypt and decrypt files while salted but could not be used with the default OpenSSL installed into Ubuntu. For example, if a file is encrypted with the standalone, OpenSSL cannot decrypt it and display the original files content.

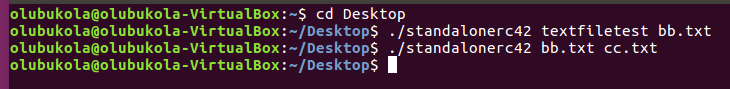
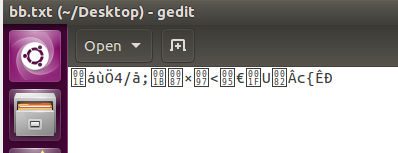
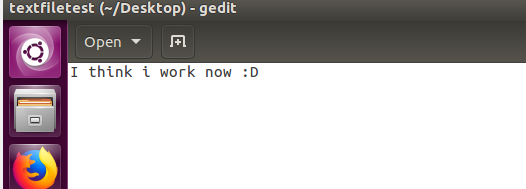


Figure 8: Command line for Encrypting and Decrypting using Standalone RC4 file



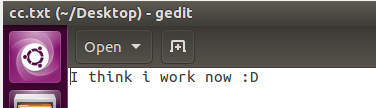


Figure 9: textfiletest is encrypted (bb.txt). bb.txt is decrypted (cc.txt)

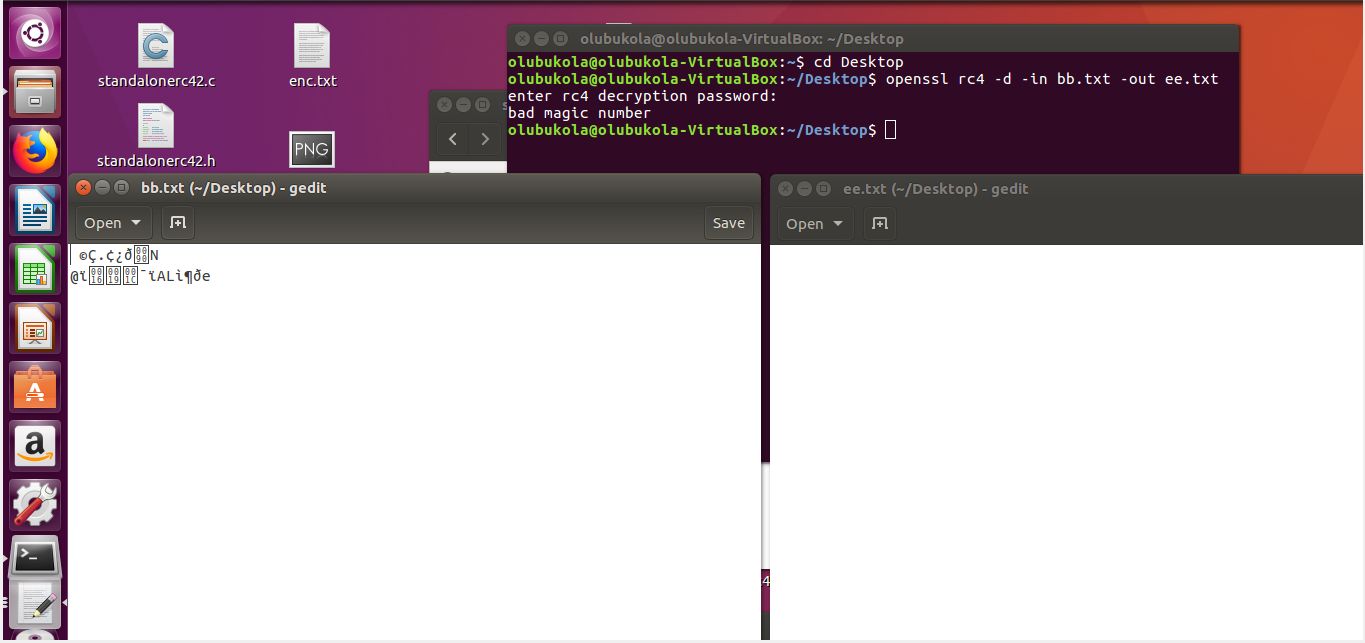


Figure 10: Unable to decrypt using Openssl

While creating the standalone RC4 utility file, I ran into two major problems. The first problem was that the linker could not locate the functions “RC4” and “RC4\_set\_key” for my standalonerc42.c source file, which caused a linker error. The solution to this problem was to include the path of the library for openssl as well as its library names. The following command line allowed for standalonerc42.c to compile successfully: gcc standalonerc42.c -I /usr/include/openssl -o standalonerc42 -lssl -lcrypto.

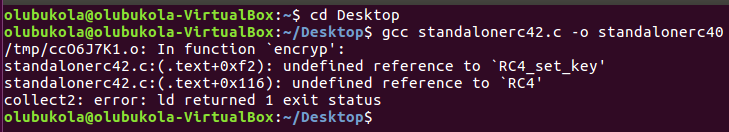


Figure 11:Incorrect command line for compiling standalonerc42.c

The second problem was that the standalone code was causing an error in mounting the mount point (i.e., Error Splicing File: Transport endpoint is not connected). The solution for this error was to relocate the encryption function call in the source file fdcache.c and to change the parameters of the read function in the standalonerc42.c source file.

Implementation Details

The following highlighted lines have been added to an existing source/header file or created with a new file:

**Standalone RC4 utility file**

**Explanation:** In the Standalonerc42.c source file, the main function takes two parameters. The first parameter is the number of arguments typed into the console and the second parameter is an array of the arguments typed into the console (i.e. argv[0] would be ./standalonerc42). The number of arguments is checked, if it is not equal to the expected number of arguments (in this case 2), the console would print an error. Next, both the input file and the output files are opened using the function “open”. The output file, since it does not exist, includes the O\_CREAT flag. This flag creates a new document if it has not been created. Afterwards, the code checks if opening the two files have failed, if so, it returns an error. The input and output file is then passed on to the encryp function. After the encryp function has been called and completed its tasks, the two files are then closed.

In the encryp function, it takes two arguments as its parameters. The first argument is the input file and the second is the output file. Two variables are then created, one of type RC4\_KEY and the other of type off\_t. The RC4\_KEY variable will be used to hold the key created from a user specified password. The off\_t variable will be used to hold the size of the input file in bytes. The function lseek is used to get the size of the file. lseek takes the input file (fd) and reads until the end of the file (SEEK\_END). The result is stored in the off\_t variable. Next, the off\_t variable is checked, if it is less than 0 then the size of the input file could not be provided and it returns an error. lseek is used again to reset the byte counter to the beginning of the input file (using SEEK\_SET). In the next two lines, the variables inbuff and outbuff is created as the buffers of the input and output file, respectively. Both are initialized using the off\_t variable. The input file is read using the read function and checked if the results return -1. If the result returns -1, then the file was not successfully read. Next, two rc4.h functions are called, RC4\_set\_key and RC4. RC4\_set\_key takes the RC4\_KEY variable created earlier, the size of the password (H\_KEY, this was defined outside of the encrypt function), and the password itself. This function creates a key from the password provided and stores it in the RC4\_KEY variable. RC4 (the function) takes the RC4\_KEY, the size of the input file, the input buffer and the output buffer. This function takes the contents of the input buffer and encrypts it using the RC4\_KEY. It then stores the encrypted results in the output buffer. The contents of the output buffer are then written to the output file (fd2). Once the content has been successfully written to the output file, both buffers are deallocated.

**Standalonerc42.h**

#ifdef STANDALONE\_RC42\_H

#define STANDALONE\_RC42\_H

int encryp(int fd, int fd2);

#endif

**Standalonerc42.c**

#include <stdio.h>

#include <stdlib.h>

#include <openssl/rc4.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#include "standalonerc42.h"

//#include <rc4\_skey.c>

//#include <rc4\_enc.c>

//Note to self: its bad practice to #include a source file :(

//#include <rc4\_locl.h>

#define H\_KEY "yufytugjhbgytcrtxc"

int encryp(int fd, int fd2){

RC4\_KEY key;

off\_t filesize; //will hold the size of the file

filesize = lseek(fd,0,SEEK\_END); //records the size of file in bytes

if(filesize < 0){ //checks for error

perror("Cannot provide offset");

exit(0);

}

lseek(fd,0,SEEK\_SET);// set byte counter back to first byte

unsigned char \*inbuff = (unsigned char\*)malloc(filesize); //input buffer

unsigned char \*outbuff = (unsigned char\*)malloc(filesize); //output buffer

if(read(fd,inbuff,filesize) == -1){ //checks if file is readable

perror("cannot read file");

exit(0);

}

//encryption / decryption

RC4\_set\_key(&key,sizeof(H\_KEY),(const unsigned char \*)H\_KEY);

RC4(&key,filesize,inbuff,outbuff);

if(write(fd2,outbuff,filesize) == -1){

perror("cannot write to file");

exit(0);

}

free(inbuff);

free(outbuff); //free memory that was allocated for outbuff

return 0;

};

int main(int argc, char\* argv[]){

int input;

int output;

//check for the number of arguments

if(argc != 3) //8

{

perror("Incorrect number of arguments!");

// exit(0);

}

//open both files

input = open(argv[1],O\_RDONLY,0644);

if(input == -1){

perror("error opening the input file");

}

output = open(argv[2],O\_WRONLY|O\_APPEND|O\_CREAT,0644);

if(output == -1){

perror("error opening the output file");

}

//pass both files to encryption file

encryp(input,output);

//close both files

close(input);

close(output);

return 0;

};

**Integration of RC4 to S3FS**

**Explanation:** The encryp function used for the standalone RC4 utility file is the same function used for the integration of RC4 to S3FS, with the exception of the following:

* The encryp function only takes one parameter since it will output the encrypted content to the same file
* lseek (SEEK\_SET) is used again after reading to reset the position byte to 0.

In the fdcache.cpp file, I placed function calls for encryp inside of FdEntity::Load and FdEntity::RowFlush. Both files were selected because they were responsible for uploading and downloading files to the Amazon S3 bucket.

**rc4test.h**

#ifndef RC4\_TEST

#define RC4\_TEST

int encryp(int fd);

#endif

**rc4test.cpp**

#include <iostream>

#include <openssl/rc4.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include "rc4test.h"

using namespace std;

#define H\_KEY "yufytugjhbgytcrtxc"

int encryp(int fd){

RC4\_KEY key;

off\_t filesize; //will hold the size of the file

filesize = lseek(fd,0,SEEK\_END); //records the size of file in bytes

if(filesize < 0){ //checks for error

perror("Cannot provide offset");

exit(0);

}

lseek(fd,0,SEEK\_SET);// set byte counter back to first byte

unsigned char \*inbuff = (unsigned char\*)malloc(filesize); //input buffer

unsigned char \*outbuff = (unsigned char\*)malloc(filesize); //output buffer

if(read(fd,inbuff,filesize) == -1){ //checks if file is readable

perror("cannot read file");

exit(0);

}

//encryption / decryption

RC4\_set\_key(&key,sizeof(H\_KEY),(const unsigned char \*)H\_KEY);

RC4(&key,filesize,inbuff,outbuff);

//write encryption to new file

lseek(fd,0,SEEK\_SET);// set byte counter back to first byte

if(write(fd,outbuff,filesize) == -1){

perror("cannot write to file");

exit(0);

}

free(inbuff);

free(outbuff); //free memory that was allocated for outbuff

return 0;

};

**fdcache.cpp**

#include “rc4test.h”

........

.........

int FdEntity::Load(off\_t start, size\_t size)

{

S3FS\_PRN\_DBG("[path=%s][fd=%d][offset=%jd][size=%jd]", path.c\_str(), fd, (intmax\_t)start, (intmax\_t)size);

//encryp(fd);

if(-1 == fd){

return -EBADF;

}

AutoLock auto\_lock(&fdent\_lock);

int result = 0;

// check loaded area & load

fdpage\_list\_t unloaded\_list;

if(0 < pagelist.GetUnloadedPages(unloaded\_list, start, size)){

for(fdpage\_list\_t::iterator iter = unloaded\_list.begin(); iter != unloaded\_list.end(); ++iter){

if(0 != size && static\_cast<size\_t>(start + size) <= static\_cast<size\_t>((\*iter)->offset)){

// reached end

break;

}

// check loading size

size\_t need\_load\_size = 0;

if(static\_cast<size\_t>((\*iter)->offset) < size\_orgmeta){

// original file size(on S3) is smaller than request.

need\_load\_size = (static\_cast<size\_t>((\*iter)->next()) <= size\_orgmeta ? (\*iter)->bytes : (size\_orgmeta - (\*iter)->offset));

}

size\_t over\_size = (\*iter)->bytes - need\_load\_size;

// download

if(static\_cast<size\_t>(2 \* S3fsCurl::GetMultipartSize()) <= need\_load\_size && !nomultipart){ // default 20MB

// parallel request

// Additional time is needed for large files

time\_t backup = 0;

if(120 > S3fsCurl::GetReadwriteTimeout()){

backup = S3fsCurl::SetReadwriteTimeout(120);

}

result = S3fsCurl::ParallelGetObjectRequest(path.c\_str(), fd, (\*iter)->offset, need\_load\_size);

if(0 != backup){

S3fsCurl::SetReadwriteTimeout(backup);

}

}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;

}

encryp(fd);

....

....

}

int FdEntity::RowFlush(const char\* tpath, bool force\_sync)

{

int result;

S3FS\_PRN\_INFO3("[tpath=%s][path=%s][fd=%d]", SAFESTRPTR(tpath), path.c\_str(), fd);

// encryp(fd);

if(-1 == fd){

return -EBADF;

}

AutoLock auto\_lock(&fdent\_lock);

if(!force\_sync && !is\_modify){

// nothing to update.

return 0;

}

encryp(fd);

// If there is no loading all of the area, loading all area.

size\_t restsize = pagelist.GetTotalUnloadedPageSize();

......

......

}

Future Improvement

During the time allocated to do this project, I was not able to fully implement my standalone RC4 utility file. Although I could encrypt and decrypt using the standalone by itself, it did not work with OpenSSL. In the future I would like the standalone to be fully functional with OpenSSL salt and OpenSSL nosalt.

Summary

This project truly challenged my abilities as a programmer and tested my knowledge of the concepts taught in class. I learned how to use new software and programs like S3FS, OpenSSL, Amazon S3 and used programs I never had the chance to explore in the past like Visual Studio Code and Virtual Box. I became more familiar and comfortable with the terminal and using commands. I also made use of numerous lines of documentation to figure out how a program would work. I also learned to ask for help when I truly needed it (Thank you, Dr. Xu, Sumukhi and Rui!). This has been a very difficult project for me to complete, but I am glad I stuck with it and made progress. I have grown as a programmer because of this project and the class.