**INTRODUCTION**

Data storage is something that any and all people worry about from time to time. From running out of room for your pictures to making sure an important piece of data is saved and accessible for work. In the recent years there has been a rise to cloud server storage of individual and comapny data, where instead of holding that data locally in the machine it will be uploaded to a server somewhere offsite where the data can be accessed over the internet. This makes it so that people will not have to worry about space limitations on their local devices and that all the data they create is safely stored. The issue now is that since the data will be accessed through the internet will it be safe? Because the data will be accessed through standard internet means (i.e. web login) there is a chance that people can break through that security and steal or damage your data. This can be handled if you were to have a client program that is stored locally and when manipulating the server data makes the data only visible on the client side and the server side holds encrypted data.

There is a program called s3fs-fuse that is used to connect to s3 based server systems and access the data located within a specified bucket on the server. It works quite well wherein after running the program with the proper security credentials you will be able to see a local folder that is linked through the internet to your web based server bucket and will be able to freely add and delete files within the server as if they were stored locally. This program however simply links the host to the server and does nothing to files while they are being sent to and pulled from the server. This means that if you add a file to the local server directory then the file that would be on the server is the same not encrypted and not protected.

What can be done to remedy this and make the files more secure is the development of an encryption program that can be integrated into the s3fs program so that files will be encrypted when added to the server and decrypted when dragged into the local directory of the bucket. This project seeks to tackle this issue and provide a viable secure s3fs with encryption implementation.

**PROJECT GOALS**

There are several goals that need to be accomplished. First what is needed is a client program that will be able to connect to the amazon S3 server and be able to host local access to the server bucket. Next an encryption program will need to be developed that can encrypt and decrypt the data without damaging the files. Finally the program will need to be integrated into the client program so that whena user adds a file to the server they can see it locally but when the file is accessed from the web client it is encrypted and unreadable.

**SYSTEM INFORMATION**

CPU: Intel core i7 7700HQ

RAM: 16GB

OS: Ubuntu 16.04 LTS

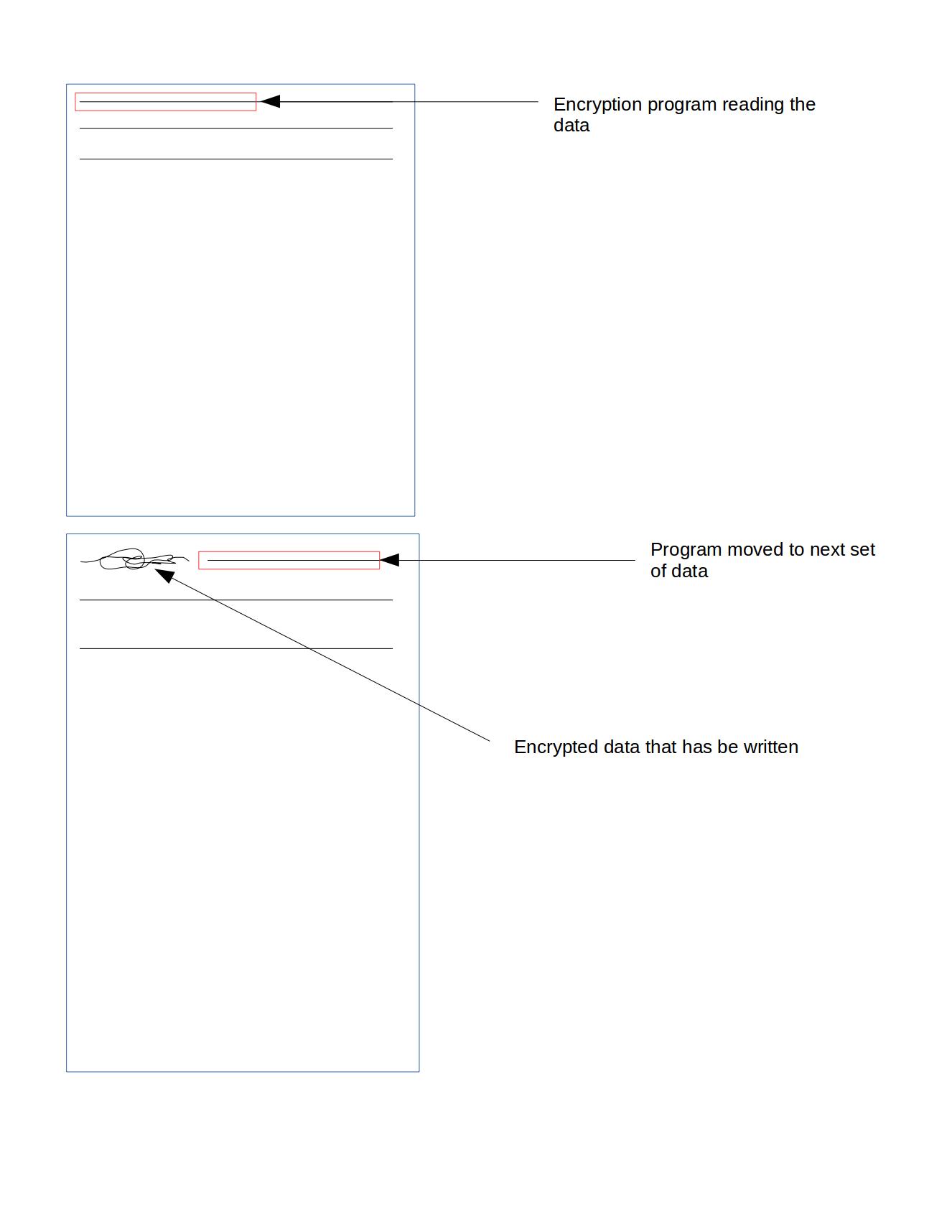
S3FS-FUSE Version: 1.80

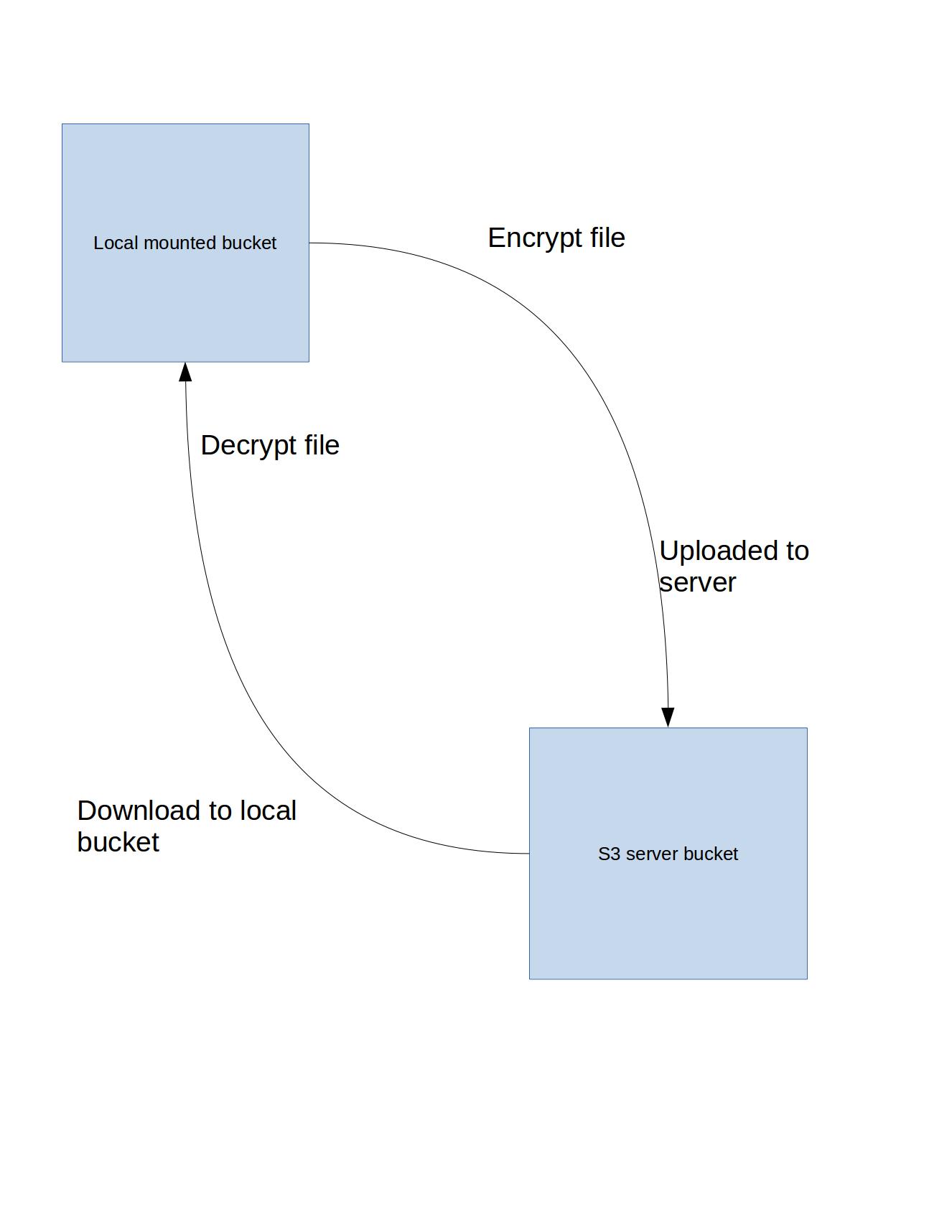
**TOOLS AND PACKAGES**

The packages that were used were s3fs-fuse for the bucket mounting client. For the encryption the MD5 and RC4 tools were used from the OpenSSL package. A total list of the included files are included in the RC4 enc folder of the s3fs-fuse package and will contain all the files used to write the encryption functions used. Looking through the files the latest version of s3fs has fuse integrated into it so the download and inclusion of individual fuse files was unnecessary.

**DESIGN**

In order to implement this program efficiently first thing would be to develop a standalone program that uses the RC4 encryption standard using a MD5 hash function for the key. This program will take in the file and then create a char array that will intake the data of the file as the program works through it. The program will encrypt the incoming data and pass the encrypted data to a unsigned char that will hold the outgoing data that will be returned. While this is going every pass the program will check to see how much data we are going to encrypt and if it is less than the max size that we predefined then it will show that we are at the end of the file and exit the program after encrypting. The way this will work in a broad sense is that the program will read a set of data, encrypt it, and then overwrite the read bytes with encrypted versions and then move to the next segment of the file.

 Once the encryption is set and the program runs it is time to design how it will integrate into s3fs-fuse. The goal of the project is to make it so that when someone adds an item locally the file in the web client is encrypted and when an encryted file is added through the web console the local bucket decrypts it automatically for the user to seemlessly access. To make sure this works there will have to be calls of the encryption function when the program loads the file on our local mounted bucket and a call when the file is being uploaded to the S3 servers.

**INTEGRATION**

This project succeded in meeting the goals of encrypting the files before loading them into the S3 server and then decrypting an encrypted file that was loaded into the web client before it is loaded into the locally mounted bucket. So when jpeg file is added to the locally mounted bucket folder on the client computer it is readable, then when trying to read the file from the web browser client it will be encrypted and undreadable. And when an encrypted file is added to the bucket using the browser web client it will stay encrypted when accessed in the browser, but when accessed from the local bucket it is decrypted and readable for the user. This was the goal and it has met that goal.

The standalone rc4 encryption program also met the goal of encrypting files. It will take a file and encrypt it and then when you run the program again on the encrypted file it will decrypt it. When running the diff function on the resulting file and the original it indicates that they are identical.

However, there are some areas where the program does fall short of the goals, mainly with openssl. Ideally the programs that were written should be compatible with the openssl version of the rc4 encryption, meaing that any file that is encrypted using the written program will be decrypted by the openssl version and vice versa. In the end it is seen the when using the custom encryption to encrypt, if the file is larger than 4kb, then the open ssl program wont decrypt past a certain point in the file. This is frustrating because included in the source file for the custom encryption is a version of the encryption that works with the files and openssl, however in order to integrate the program into s3fs two new function were created, one that would generate the key in char form and another that would call that function on a char buffer obtained from a fd object. The new functions integrated into s3 perfectly but lead to the issue of having open ssl not being to fully decrypt a file that was encrypted using the new functions.

**IMPLEMENTATION DETAILS**

First thing that was implemented was the RC4 standalone. To get this to work what needed to be done was the writing of the functions that would encrypt and decrypt a file and the main function that would call those function to encrypt. To that end many files from the openssl program were needed to get the rc4 that was written to work in a similar fashion. Then a key, which is used to make the key, was defined along with a length variable of the key were defined as global variables. Then there is a function that was made where the file is converted to a string and then md5 is used to hash the password and stored into an object. The object is then used to set the key and the RC4 encryption function is called to encrypt using the information we have provided in the statements before it. Finally it will return the encoded data. This however was not enough to work with s3fs and so two different functions were made. One that worked very similar to the previous function however of type unsigned char and another that takes a fd object as a parameter. The new encrypt function works the same way as the string version though with slight modifications to account for the new data type. The final function takes in the fd object and then creates a buffer of 4096 bytes and an char array of that size. Will then assign the size of the file to a variable and starting from the beginning start encrypting and replacing data 4096 bytes at a time. Every iteration a value is incremented to keep track of how much data is about to be processed, if the value is ever less than the size of the buffer it will encrypt that data and then end the program.

RC4 standalone main implementation:

This includes the main function to allow this to be called as a stand alone.

#include "rc4\_enc.c"

#include "rc4.h"

#include "rc4\_skey.c"

#include "rc4\_locl.h"

#include <iostream>

#include <string>

#include <fstream>

#include <sstream>

#include <cstdlib>

#include <stdio.h>

#include <fcntl.h>

#include "md5.h"

using namespace std;

//definition of the key variables for the program to use to make the encryption

#define RC4KEY "pavanisgreat"

#define RC4LEN (sizeof(RC4KEY) - 1)

//function prototypes

string encrypt(const string &input);

unsigned char\* encryptChar(const char\* input, int size);

void encdec(int fd);

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

{

/\*

//turning the file data into binary for encryption

ifstream infile;

infile.open(argv[1], ios::binary);

stringstream test;

test << infile.rdbuf();

string temp = test.str();

string temp1 = encrypt(temp);

infile.close();

//storing the ecrypted data into a new file.

ofstream out;

out.open(argv[2], ios::binary);

out << temp1;

stringstream ststream;

ststream << out.rdbuf();

string temp2 = ststream.str();

out.close();\*/

int fd;

fd = open(argv[1], O\_RDWR);

encdec(fd);

}

//function that will encrypt a file after the file data has been converted to a string object.

string encrypt(const string &input)

{

RC4\_KEY rc4;

int length = input.size();

//char variable that will hold the data for the actual encryption

unsigned char \*indata = (unsigned char \*)malloc(length + 1);

//setting the data to 0

memset(indata, 0, length + 1);

//creating the char variable that will hold encryption information based on key variables and the MD5 standard

unsigned char \*md[MD5\_DIGEST\_LENGTH];

MD5((unsigned char\*)RC4KEY, RC4LEN, (unsigned char\*)&md);

RC4\_set\_key(&rc4, MD5\_DIGEST\_LENGTH, (const unsigned char \*)md);

//encrypting the file

RC4(&rc4, length, (const unsigned char \*)input.c\_str(), indata);

//creating the string object that will hold the encrypted data

string encoded((char \*)indata, length);

free(indata);

return encoded;

}

//encryption function that works on unsigned char rather than string which will make it easier to encrypt based on a file descriptor object

unsigned char\* encryptChar(const char\* input, int size)

{

RC4\_KEY rc4;

int length = size;

unsigned char \*indata = (unsigned char \*)malloc(length + 1);

memset(indata, 0, length + 1);

unsigned char md[MD5\_DIGEST\_LENGTH];

MD5((unsigned char\*)RC4KEY, RC4LEN, (unsigned char \*)md);

//cout << "!!!!!!!!!!! RC4KEY AND LENGTH !!!!!!!!!!! " << RC4LEN << " " << RC4KEY << endl;

//cout << "!!!!!!!!!!! MD Before !!!!!!!!!!!!!" << md << endl;

RC4\_set\_key(&rc4, MD5\_DIGEST\_LENGTH, (const unsigned char \*)md);

//cout << "!!!!!!!!!!! MD after !!!!!!!!!!!!!" << md << endl;

RC4(&rc4, length, (const unsigned char \*)input, indata);

unsigned char\* encoded;

encoded = indata;

return encoded;

}

// the function that will be called by S3FS to encrypt the file that is being uploaded

void encdec(int fd)

{

//setting the variables to work on fd obejects

const int bufferSize = 4096;

char incoming[bufferSize];

unsigned char\* outgoing;

//setting the length to the size of the file

int len = lseek(fd, 0, SEEK\_END);

//reseting the program to the beginning of the file

lseek(fd, 0, SEEK\_SET);

//variables to keep track of how much data has been encrypted and how much is about to be encrypted.

int found = 0;

int changed = 0;

//loop that will go through the file and encrypt it in parts.

while(len>0)

{

//checking to see how much data is left

if (len>=bufferSize)

{

changed=bufferSize;

len = len-bufferSize;

}

else

{

changed =len;

len=0;

}

//reading the part of the data that is being grabbed

pread(fd, incoming, changed, found);

//encrypting that data

outgoing = encryptChar(incoming, changed);

//writing that data to the file

pwrite(fd, outgoing, changed, found);

//incrementing the amount of data that has been encrypted

found= found+changed;

}

}

The following are the changes made to allow it to work as part of s3fs:

#include "rc4\_enc.c"

#include "rc4.h"

#include "rc4\_skey.c"

#include "rc4\_locl.h"

#include <iostream>

#include <string>

#include <fstream>

#include <sstream>

#include <cstdlib>

#include <stdio.h>

#include <fcntl.h>

#include "md5.h"

using namespace std;

//definition of the key variables for the program to use to make the encryption

#define RC4KEY "pavanisgreat"

#define RC4LEN (sizeof(RC4KEY) - 1)

//function prototypes

string encrypt(const string &input);

unsigned char\* encryptChar(const char\* input, int size);

void encdec(int fd);

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

{

/\*

//turning the file data into binary for encryption

ifstream infile;

infile.open(argv[1], ios::binary);

stringstream test;

test << infile.rdbuf();

string temp = test.str();

string temp1 = encrypt(temp);

infile.close();

//storing the ecrypted data into a new file.

ofstream out;

out.open(argv[2], ios::binary);

out << temp1;

stringstream ststream;

ststream << out.rdbuf();

string temp2 = ststream.str();

out.close();\*/

int fd;

fd = open(argv[1], O\_RDWR);

encdec(fd);

}

//function that will encrypt a file after the file data has been converted to a string object.

string encrypt(const string &input)

{

RC4\_KEY rc4;

int length = input.size();

//char variable that will hold the data for the actual encryption

unsigned char \*indata = (unsigned char \*)malloc(length + 1);

//setting the data to 0

memset(indata, 0, length + 1);

//creating the char variable that will hold encryption information based on key variables and the MD5 standard

unsigned char \*md[MD5\_DIGEST\_LENGTH];

MD5((unsigned char\*)RC4KEY, RC4LEN, (unsigned char\*)&md);

RC4\_set\_key(&rc4, MD5\_DIGEST\_LENGTH, (const unsigned char \*)md);

//encrypting the file

RC4(&rc4, length, (const unsigned char \*)input.c\_str(), indata);

//creating the string object that will hold the encrypted data

string encoded((char \*)indata, length);

free(indata);

return encoded;

}

//encryption function that works on unsigned char rather than string which will make it easier to encrypt based on a file descriptor object

unsigned char\* encryptChar(const char\* input, int size)

{

RC4\_KEY rc4;

int length = size;

unsigned char \*indata = (unsigned char \*)malloc(length + 1);

memset(indata, 0, length + 1);

unsigned char md[MD5\_DIGEST\_LENGTH];

MD5((unsigned char\*)RC4KEY, RC4LEN, (unsigned char \*)md);

//cout << "!!!!!!!!!!! RC4KEY AND LENGTH !!!!!!!!!!! " << RC4LEN << " " << RC4KEY << endl;

//cout << "!!!!!!!!!!! MD Before !!!!!!!!!!!!!" << md << endl;

RC4\_set\_key(&rc4, MD5\_DIGEST\_LENGTH, (const unsigned char \*)md);

//cout << "!!!!!!!!!!! MD after !!!!!!!!!!!!!" << md << endl;

RC4(&rc4, length, (const unsigned char \*)input, indata);

unsigned char\* encoded;

encoded = indata;

return encoded;

}

// the function that will be called by S3FS to encrypt the file that is being uploaded

void encdec(int fd)

{

//setting the variables to work on fd obejects

const int bufferSize = 4096;

char incoming[bufferSize];

unsigned char\* outgoing;

//setting the length to the size of the file

int len = lseek(fd, 0, SEEK\_END);

//reseting the program to the beginning of the file

lseek(fd, 0, SEEK\_SET);

//variables to keep track of how much data has been encrypted and how much is about to be encrypted.

int found = 0;

int changed = 0;

//loop that will go through the file and encrypt it in parts.

while(len>0)

{

//checking to see how much data is left

if (len>=bufferSize)

{

changed=bufferSize;

len = len-bufferSize;

}

else

{

changed =len;

len=0;

}

//reading the part of the data that is being grabbed

pread(fd, incoming, changed, found);

//encrypting that data

outgoing = encryptChar(incoming, changed);

//writing that data to the file

pwrite(fd, outgoing, changed, found);

//incrementing the amount of data that has been encrypted

found= found+changed;

}

}

To integrate the encryption into the s3fs program changes to the source code were needed. First the folder that contained the standalone program were added to the s3fs directory, and then from there the rc4 main file was edited to work as a part of a larger program rather than a standalone. Because the upload and download functionality was handled in the fdchache file that was the file that was edited to integrate. In order to ensure the integration worked first the main file from the rc4 program was included in the headers. Next the download function was identified as load and there a call to the encryption function was made before the file was actually returned and then loaded into the directory. Next the upload function was identified as rowflush and here the function was called right before the file was locked and then uploaded to the server.

Fdcache.cpp edits:

inclusion of the rc4 file that contained the encryption functions

#include "RC4enc/main.cpp"

function call in the Load function before the object is returned.

encdec(fd);

Function call in the Rowflush function after the file is identified and before the auto-lock happens for the upload.

encdec(fd);

**FUTURE IMPROVEMENT**

There are a few things that can be done to further improve on the current program. First off would be to fix the issue of the openssl encryption not working completely on the current implementation of the s3fs encryption. Another improvement that would be nice to see is a check to make sure that if a file is uploaded on the web client un-encrypted it will not encrypt itself when loaded to the local bucket. Another possible improvement would be to use a better encryption method. Though RC4 is easy to work with and does encrypt a file it is not the most secure encryption out there. Even with the integration of MD5 hashing, other encryptions like AES can be more secure and powerful.

**SUMMARY**

This was a very rewarding a fulfilling project to take on. From the project I was able to learn about how encryption programs worked and how they go about protecting data. I was also able to learn about how the base encryption programs can be altered to increase security and make it harder for intruders to harm any of my data. In addition to encryption I learned a great deal about web based storage programs and the local clients that many of them include. Looking through the source code gave me insight into how the local clients read, interpret, store and deal with data between a local directory and the actual server where the information is stored. Though the project is over I feel there is more I can do and as I learn more hope to come back to this project to improve its functionality.