Name: Anush Rajam

Due Date: May 4th, 2021

Class: CSC 4420

Assignment: Final Project

Contents

[Introduction 3](#_Toc70855961)

[Project Goal 3](#_Toc70855962)

[System Information 3](#_Toc70855963)

[Tools and Packages 4](#_Toc70855964)

[Implementation Details 4](#_Toc70855965)

[Design 9](#_Toc70855966)

[Integration 9](#_Toc70855967)

[Future Improvement 10](#_Toc70855968)

[Summary 11](#_Toc70855969)

[Screenshots 11](#_Toc70855970)

## Introduction

FUSE: a userspace filesystem framework. It consists of a kernel module (fuse.ko), a userspace library (libfuse.\*) and a mount utility (fusermount). FUSE allows secure, non-privileged mounts. This opens up new possibilities for the use of filesystems.

RC4: Rivest Cipher 4, or RC4, is a stream cipher created in 1987. A stream cipher is a type of cipher that operates on data a byte at a time to encrypt that data. RC4 is extremely simple to use, thus making the implementation simple as well. RC4 also works with large streams of data swiftly and easily.

S3FS: 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. S3FS will mount an amazon s3 bucket (that has been properly formatted) as a local file system.

This project takes an Amazon S3-based FUSE implementation and wedges a RC4 enryption layer between the upload and download stages of synchronization.

## Project Goal

In this project, you will design and implement a stackable file system running on Linux, which uses Amazon's S3 as backend storage. The following requirements should be met:

* Once mounted, a bucket in S3 appears as a local directory on your Linux machine;
* Any updates ( creation, deletion or modification of files and subdirectories ) in the local directory are reflected in the corresponding S3 bucket, and vice vera;
* A stand-alone rc4, fully compatible with "openssl rc4", with both nosalt and salt options, i.e., any files encrypted by the stand-alone rc4 should be able to be decrypted by "openssl rc4", and vice versa;
* all files in the file system are automatically encrypted using rc4 with salt, and applications can transparently operate on the files without explicit decryptions.

## System Information

**Kernel**



**CPU**

Text

Description automatically generated





|  |  |
| --- | --- |
| Software | Version |
| libfuse | 3.2.2 |
| s3fs-fuse | 2.9.9 |
| openssl | 1.1.1f |

## Tools and Packages

The following packages and tools are used in this project:

* libfuse (FUSE reference implementation library)
* s3fs-fuse (Amazon S3 FUSE implementation)
* openssl (Encryption library used for RC4 encryption and decryption)

## Implementation Details

* RC4 Implementation code:

#include <openssl/rc4.h>

#include <openssl/evp.h>

#include <openssl/rand.h>

#include <stdio.h>

#include <stdbool.h>

#include <unistd.h>

#include <string.h>

#include <fcntl.h>

#define SALT\_SIZE 8

#define KEY\_SIZE 16

#define SALT\_BUF\_SIZE 16

#define BUFF\_SIZE 4096

/\*

Usage: ./main [-e | -d] [-salt | -no-salt] [key] [input file] [out\_file]

-e - encrypt or -d - decrypt

\*/

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

int inFile;

int outFile;

unsigned char transformedKey[KEY\_SIZE];

RC4\_KEY key;

char in\_Buf[BUFF\_SIZE];

char out\_Buf[BUFF\_SIZE];

if (argc < 6){

printf("Error: Missing arguments.\n");

printf("Format: ./main [-e | -d] [-salt | -no-salt] [key] [input file] [out\_file]\n");

exit(-1);

}

else if (argc > 6){

printf("Error: Excessive arguments.\n");

printf("Format: ./main [-e | -d] [-salt | -no-salt] [key] [input file] [out\_file]\n");

exit(-1);

}

if (! (strcmp(argv[1], "-e") == 0 || strcmp(argv[1], "-d") == 0) ){

printf("Error: Incorrect [-e | -d] input.\n");

printf("Format: ./main [-e | -d] [-salt | -no-salt] [key] [input file] [out\_file]\n");

exit(-1);

}

if (! (strcmp(argv[2], "-salt") == 0 || strcmp(argv[2], "-no-salt") == 0) ){

printf("Error: Incorrect [-salt | -no-salt] input.\n");

printf("Format: ./main [-e | -d] [-salt | -no-salt] [key] [input file] [out\_file]\n");

exit(-1);

}

if (access(argv[4], F\_OK) != 0){

printf("Error: Input File does not exist.\n");

printf("Format: ./main [-e | -d] [-salt | -no-salt] [key] [input file] [out\_file]\n");

exit(-1);

}

// opens file for reading

inFile = open(argv[4], O\_RDONLY);

// opens file for create and write, gives most permissions over file to current user

outFile = open(argv[5], O\_CREAT | O\_WRONLY, 0644);

if (strcmp(argv[2], "-salt") == 0){

// SALT

unsigned char salt[SALT\_SIZE];

if (strcmp(argv[1], "-e") == 0){

// Encryption

char saltBuf[SALT\_BUF\_SIZE];

memset(saltBuf, 0, SALT\_BUF\_SIZE);

RAND\_bytes(salt, SALT\_SIZE);

sprintf(saltBuf, "Salted\_\_%c%c%c%c%c%c%c",

salt[0], salt[1], salt[2], salt[3],

salt[4], salt[5], salt[6]);

saltBuf[15] = salt[7];

write(outFile, &saltBuf, SALT\_BUF\_SIZE);

}

else if (strcmp(argv[1], "-d") == 0){

// Decryption

lseek(inFile, 8, SEEK\_SET);

read(inFile, salt, SALT\_SIZE);

}

// SALT KEY

if (! EVP\_BytesToKey(EVP\_rc4(), EVP\_sha256(),

salt, (unsigned char \*)argv[3], strlen(argv[3]), 1, transformedKey, NULL) )

{

printf("Error: Could not create encryption key.\n");

exit(-1);

}

RC4\_set\_key(&key, KEY\_SIZE, transformedKey);

}

else {

// NO SALT KEY

if (! EVP\_BytesToKey(EVP\_rc4(), EVP\_sha256(),

NULL, (unsigned char \*)argv[3], (int)strlen(argv[3]), 1, transformedKey, NULL) )

{

printf("Error: Could not create encryption key.\n");

exit(-1);

}

RC4\_set\_key(&key, KEY\_SIZE, transformedKey);

}

// WRITE TO FILE

ssize\_t bytesRead = 0;

while (bytesRead = read(inFile, &in\_Buf, BUFF\_SIZE)) {

RC4(&key, bytesRead, (const unsigned char\*) in\_Buf, (unsigned char\*) out\_Buf);

write(outFile, &out\_Buf, bytesRead);

}

// handle closing files

close (inFile);

close (outFile);

return 0;

}

* S3FS Implementation code:
  + Located in CSC4420\_S3FS\_RC4/src/fdcache\_entity.cpp

//Encryption/decryption headers

#include <openssl/rc4.h>

#include <openssl/evp.h>

#include <openssl/rand.h>

#include <stdio.h>

#include <stdbool.h>

#include <unistd.h>

#include <string.h>

#include <fcntl.h>

# define NUM\_OF\_SALT 8

# define NUM\_OF\_KEYS 16

# define SALT\_STR\_LEN 16

# define READ\_WRITE\_SIZE 4096

//key file location

#define keyLocation "/home/anush/Documents/key.txt"

#define tempLocation "/home/anush/Documents/temp.txt"

//function to grab key from key.txt. Return number of characters

void grabKey(int file, char \*key, int size){

int i=0;

char buf[size];

while (read(file, (char \*)&buf[i++], 1)); //read key into buf

for (int i=0; i<size; i++){

key[i] = (char)buf[i]; //read buf into key var

}

close (file); //close file

}

void runRC4(int inFile, char \*type){

unsigned char transformed\_key[NUM\_OF\_KEYS];

RC4\_KEY key;

unsigned char inBuff[READ\_WRITE\_SIZE];

unsigned char outBuff[READ\_WRITE\_SIZE];

unsigned char salt[NUM\_OF\_SALT];

const char \*e = (char \*)"e";

const char \*d = (char \*)"d";

lseek(inFile, 0, SEEK\_SET); //make sure were reading from beginning of file

int outFile = open(tempLocation, O\_CREAT | O\_RDWR, 0664); //rw-rw-r-- temp file to read file into during upload. allows us to add saltstring to inFile.

//if encrypting

if (strcmp(type, e) == 0){

char saltBuf[SALT\_STR\_LEN];

memset(saltBuf, 0, SALT\_STR\_LEN); //set saltBuf to all zeros before reading into it

RAND\_bytes(salt, NUM\_OF\_SALT); //randomly generate salt bytes

sprintf(saltBuf, "Salted\_\_%c%c%c%c%c%c%c", //string to store salt in ciphertext

salt[0], salt[1], salt[2], salt[3],

salt[4], salt[5], salt[6]);

saltBuf[15] = salt[7]; //simple solution to fix salt[7] not being printed in properly

write(outFile, &saltBuf, SALT\_STR\_LEN); //write saltBuf to output file

}

//if decrypting

else if (strcmp(type, d) == 0){

lseek(inFile, 8, SEEK\_SET); //read/write file offset to read saltBuf

read(inFile, salt, NUM\_OF\_SALT); //read salt chars into salt

}

//now that we have the salt generated or read, we can create key with salt

int size = 0;

char \*regular\_key;

//if file cannot be found, set key to default: fi1942 of size 6.

if (access(keyLocation, F\_OK) != 0){

printf("Error: Could not find key.txt. Using default key instead.\n");

size = 6;

regular\_key = (char \*)malloc(size \* sizeof(unsigned char \*));

sprintf(regular\_key, "fi1942"); //set key to default: fi1942

}

//if file does exist read key from it

else{

int file = open(keyLocation, O\_RDONLY); //try to open key file

size = lseek(file, 0, SEEK\_END) - 1; //scan key file to check for size

lseek(file, 0, SEEK\_SET); //reset key file pointer back to beginning

regular\_key = (char \*)malloc(size \* sizeof(unsigned char \*));

grabKey(file, regular\_key, size); //grab key from key.txt

}

//generate key with salt and regular\_key

if (! EVP\_BytesToKey(EVP\_rc4(), EVP\_sha256(),

salt, (unsigned char \*)regular\_key, size, 1, transformed\_key, NULL) )

{

printf("Error: Could not create encryption key.\n"); //if BytesToKey fails.

exit(-1);

}

RC4\_set\_key(&key, NUM\_OF\_KEYS, transformed\_key); //using transformed\_key hash to create RC4\_KEY key.

//write data to outFile. using while loop to only enncrypt & write by READ\_WRITE\_SIZE bytes at a time

ssize\_t readBuff = 0; //counting bytes so program knows what buffer is if buffer < READ\_WRITE\_SIZE

if (strcmp(type, e) == 0){

while (readBuff = read(inFile, &inBuff, READ\_WRITE\_SIZE)){

RC4(&key, readBuff, inBuff, outBuff); //encrypt inBuff to outBuff

write(outFile, &outBuff, readBuff);

}

//after writing inFile to outFile, we now have an outFile of saltBuf + inputfile.

lseek(outFile, 0, SEEK\_SET); //reset read pointer

lseek(inFile, 0, SEEK\_SET); //reset read pointer

while (readBuff = read(outFile, &outBuff, READ\_WRITE\_SIZE)){ //write outFile with saltstring back to inFile

write(inFile, &outBuff, readBuff);

}

}

else if (strcmp(type, d) == 0){

int offset = 0;

while (readBuff = read(inFile, &inBuff, 16)){ //16 = max size we can have to overwrite as were reading

//(size of saltBuf). This allows us to continually read and write from same file.

RC4(&key, readBuff, inBuff, outBuff); //encrypt inBuff to outBuff

pwrite(inFile, &outBuff, readBuff, offset);

offset += readBuff;

}

ftruncate(inFile, offset); //truncate file size to remove unneeded 16 bytes from deleted saltedString.

}

remove(tempLocation); //delete temp.txt file we created earlier

}

## Design

Diagram

Description automatically generated

## Integration

* Accomplished:
  + Installation:
    - Installed Virutal Box + Linux Image
    - Installed all packages
      * FUSE
      * S3FS
      * GCC-C++
      * Openssl
  + AWS:
    - Created AWS Account, retrieve AWS credentials
    - Created S3 Bucket
    - Created AWS credential file in Ubuntu to configure S3FS
    - Successfully used S3FS terminal functions to mount S3 bucket to a specified directory
    - Tested file upload to mounted directory
  + RC4:
    - Successfully created RC4 standalone script in C
  + RC4 Testing:
    - Tested our encryption and decryption rc4 script using the OpenSSL command line tool
      * Encrypted with the RC4 script, decrypted with the OpenSSL command line tool vice versa.
  + SALT:
    - Successfully integrated SALT into RC4 Script
  + SALT Testing:
    - Tested encryption and decryption with our RC4 Script
* What’s not accomplished:
  + We integrated RC4 encryption and decryption to the write and load functions in S3FS source code, however it never worked.
* What went wrong:
  + Installation:
    - S3FS installation errors and incompatibilities
      * We first used windows subsystem but realized many needed drivers were missing
    - S3 bucket upload issues arising from bad s3Fs install
    - Found an error with mounting via -f flag (which gives verbose output)
      * Multiple issues with S3FS process of uploading and syncing with AWS S3

## Future Improvement

Finish integration of S3FS with the RC4 script successfully. Additionally, Updating the encryption algorithm to something more recent in OpenSSL. Further improvements can be made to the encryption algorithm such as using Blowfish, which is another encryption algorithm in OpenSSL. Although at this time I do not know much about it, many articles online suggested using Blowfish because it is safer to use then RC4.

## Summary

Overall, the project was very informative. It challenged me to use a library I’ve never seen and read/use functions within it. Also, it taught me how to contribute to open source projects, an important skill that I have been planning to get into as a future software engineer.

In the beginning the project started out very straightforward, however as the project went along there were numerous errors that came up. The most challenging aspect of this project was to find solutions for each error that came up. I am also grateful to have gotten the opportunity to develop in a Linux environment and learn the Linux terminal.

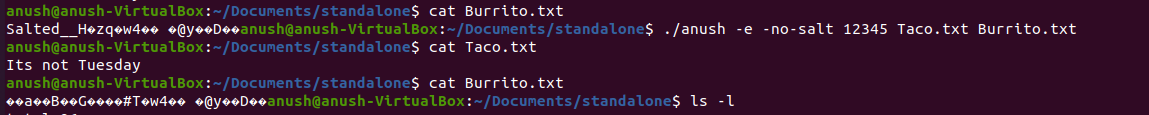
## Screenshots

Salted encryption:

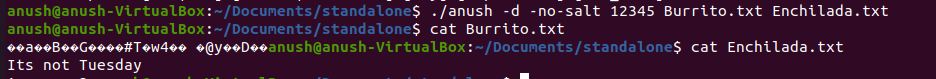
Text

Description automatically generated

No Salt Encryption:



No Salt Decryption:



Salt Decryption:

