CSC 4420 Final Project Report

Adrian Ionascu

April 30, 2017

1 Introduction

With "The Cloud" being a more prominent idea in the tech industry, it comes with drawbacks and issues. More and more people use backups and send them to cloud services and cloud storage, as for the average user it is easier and it can assure them some relief in case of the loss of their original data. This project comes to cover one of the ways to make cloud based storage more secure through encryption.

2 Project Goals

The goal of this project was to add new functionality to s3fs by adding RC4 encryption. This is done so that files put into the S3 mount by the user are to be encrypted before they are uploaded to the user's bucket, and the files will be decrypted upon download as needed by the user. The key for the encryption is to be specified by the user so that the user has control as to how their files are encrypted.

The RC4 encrypted to be integrated and implemented into s3fs must be compatible with OpenSSL's RC4 encryption. Meaning that a file encrypted with our s3fs RC4 implementation must be capable of being decrypted back to the original file using OpenSSL. Given a file encrypted using RC4 through OpenSSL, our s3fs RC4 implementation should also be capable of decrypting it back to it's original file. The RC4 code to be implemented into s3fs must be capable of encrypting and decrypting files by blocks of data instead of being all encrypted/decrypted at once.

3 System Information

The entirety of the project was done and tested on a laptop running a VM through VM Ware.
Host OS: Windows 8.1
Guest OS: Ubuntu 16.04
CPU: Intel i5-5200U @ 2.20 GHz
GPU: Intel HD Graphics 5500
HDD: 1 TB 5400 RPM (20 GB given to guest OS)
Memory: 16 GB (8 GB given to guest OS when running)

4 Tools and Packages

S3FS - Needed to view source code and integrate encryption

OpenSSL - Needed to view source code and see how RC4 encryption is done, and was used for RC4_set_key, RC4, and MD5 functions for implementation into the project

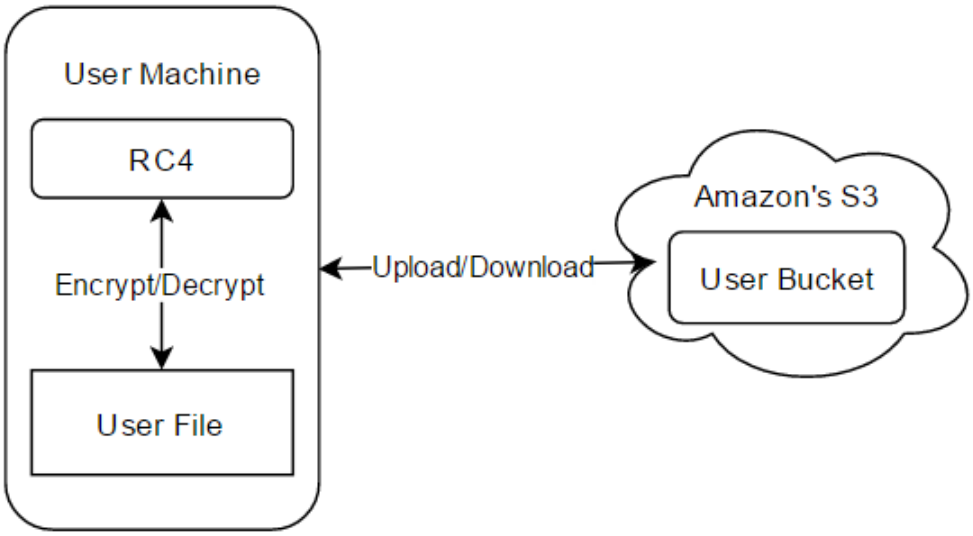
Vim/Atom - Used for text editing

g++/gdb - Used for compiling and debugging standalone encryption

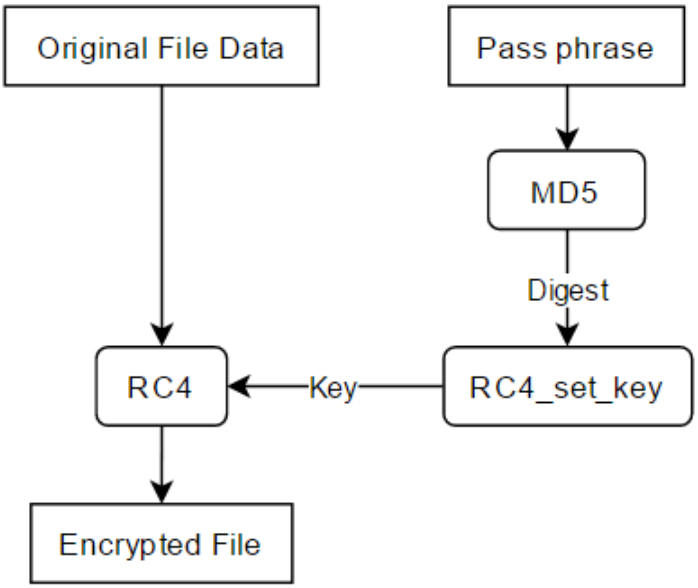
Bash - used to script command line arguments so as to mount and remount quickly during proper testing (script is included with source code)

5 Design

The design of the project is so that the users file encrypts just before upload and decrypts just after download. This is to ensure the user can still see the data as needed but also makes sure that the backup uploaded into the user's bucket on Amazon's S3 is always encrypted.



The RC4 encryption itself takes in the pass phrase and hashes it using MD5, this generates a digest of 32 hexadecimal characters (16 bytes long) and puts it through the RC4_set_key function. This function sets a key for use in the RC4 functions itself, which takes the original file data and the key and outputs an encrypted file.



6 Issues & Integration

The code integrated into s3fs works properly to encrypt uploaded files and decrypt downloaded files. A feature that was added later was the ability to specify certain command line arguments to allow the user to specify when they want s3fs to encrypt/decrypt files, as well as optional command line arguments to specify how they want to input their pass phrase for use in encryption/decryption, but not without problems in trying to integrate and implement it.

When integrating the RC4 encryption into I ran into many trouble. First of which was that when I attempted to write my own code for RC4 I noticed that there were 2 ways of going about it. One of those ways was to implement it using the functions RC4_SET_KEY and RC4. The other way was to use EVP functions that are meant to be for programmers to easily implement OpenSSL encryption without needing to write their own functions, as it can lead to unsecure or incorrect code.

I first attempted to implement the EVP functions but ran into issues where the encrypted file outputted by this code ended up being 1 byte too short. I attempted to fix this for an hour or so and only received encrypted files that were 1 byte too short or 5 bytes too long. I then attempted to write RC4 using the other functions as described. From there I was able to get my test files to encrypt, but realized that the files were encrypted differently to openssl. Looking around for a description of OpenSSL functions and other information, the OpenSSL help showed a command ”-md” to describe the hashing of the passphrase. After encrypting a few test files with the ”-md” option set to a few hashing algorithms I determined OpenSSL by default uses MD5 hashing for the passphrase given.

Adding MD5 hashing was not as much of an issue as RC4. When using the MD5 function the output is called a digest, which is a 16 bytes character array. This corresponds to 32 hexadecimal

characters. This is the input for the RC4_set_key function and as such allows my standalone RC4 correctly encrypt without padding or salting when compared to OpenSSL.

7 Implementation Details

When implementing my code into s3fs I took note of where in the code the uploads and downloads started to happen. Within the s3fs code itself there are comments saying ”upload” and ”download” mentioning where the files start uploading and downloading.

After implementing my encryption I put the encryption and decryption functions (which are the same function as RC4 is symmetric) in multiple locations during testing to see what would work out best and in the end I put my encryption function both within ”fdcache.cpp” within the write and load functions. This in the end was the location of the functions as the project worked as intended for the goal of the project.After talking to professor Lihao, it was mentioned that these locations work, but are not ideal. So in the future I would like to go through and find more appropriate spots for these functions.

After implementing my encryption I went about and added new command line arguments for s3fs that will allow the user to specify how they want the encryption to work and their password. When implementing new command line arguments I went through s3fs.cpp where s3fs' main function was, and I went through the code looking for anything that allows parsing. In s3fs there is a function in the FUSE package to parse out command line arguments in a way that is much easier for s3fs to handle, this made it so that adding a new command line argument was not very difficult.

After adding the command line argument s3fs.cpp later has checks for those arguments such as if two command line arguments were specified at the same time that might conflict with one another. I implemented this check so that the user cannot specify a passphrase for enryption through the command line argument AND give a password file through command line argument at the same time. After adding the command line argument and its checks, I proceeded to add a new global variable to specify which command line argument has been given and modified my encryption function to look for this variable to know how to get the pass phrase to be fed into the MD5 function.

8 Future Improvement

One of the future improvements that I would like to implement in the future is compression of data before encryption. This will make uploading easier and take up less time overall. This also comes with looking through the source code of s3fs in its entirety and find a better suited spot for encryption and compression of data without having to use the functions within the ”fdcache.cpp” file.

Another future improvement that could be added is to change the encryption so that the user can specify the encryption they want. Along with this I would set a certain default encryption so

that even if the user is not sure of what encryption is to be used, they can insure some security. This would also go well with specifying the type of hashing the user might want, similar to how OpenSSL allows the user to specify it all.

Lastly, one of the additions I would like to make in the future is the ability to specify encryption and pass phrases across different buckets. In s3fs the user of capable of providing a password file that will contain the ID and access key of a user's bucket. If the user is in need of accessing multiple buckets at one time, the bucket name must also be specified alongside it. It would be useful if the user can specify they want different pass phrases to be used for different buckets (one for each given ID, key, and bucket input in the password file) or even different encryption. This would let the user have more control over how their data is encrypted and sent to the buckets they have access to.

9 Summary

This project has been a learning experience every step of the way. Through this project I was able to better improve my ability to go through code, such as whats found in s3fs, and better understand it even without documentation or explanations within comments. I also better learned how certain kinds of encryption and hashing works, such as block cipher encryption and MD5 hashing. I felt like I have learned a lot and ended up being proud of what I was able to accomplish in such a short amount of time given that I have never had to deal with a project with as much code as s3fs. As a result I want to continually make it better in my free time over the course of the next year.

I will also continue on my focus on security, and with this project, I have a better understanding of the encryption used within this project. With future changes and implementations to this project I will hope to get an even better understanding of the encryption algorithms that OpenSSL have to offer and how to implement them into projects that are even larger than s3fs within the future.

10 References

EVP function references
https://wiki.openssl.org/index.php/EVP
https://www.openssl.org/docs/man1.0.2/crypto/evp.html

RC4 function references
https://www.openssl.org/docs/man1.0.2/crypto/rc4.html

MD5 function references
https://www.openssl.org/docs/man1.0.2/crypto/md5.html
https://linux.die.net/man/3/md5_init