secure s3fs

Mustafa Jaber

April 27, 2020

wayne state university

Table of contents

1. Introductions ……………………………………………………………………………………………01

* What is S3FS-FUSE?

1. Project goals ……………………………………………………………………………………………01

* Secure S3FS
* Standalone Implementation

1. System ……………………………………………………………………………………………02
2. Tools and Packages ………………………………………………………………………………………02

* S3FS-Fuse
* OpenSSL
* Amazon S3

1. Design ……………………………………………………………………………………………03

* RC4

1. Integrations ……………………………………………………………………………………………04

* S3FS Integration
* Standalone Integration
* Salt –
* S3FS Standalone Integration – unfinished

1. Implementation ……………………………………………………………………………………………05

* S3FS Implementation
* Standalone Implementation
* S3FS Standalone Implementation

1. Future Improvement ……………………………………………………………………………………07

* Salt
* Integration

1. Summary ……………………………………………………………………………………………10

* Accomplishments
* What I learned

1. Closing ……………………………………………………………………………………………10

1. **Introduction**

My name is Mustafa Jaber and is a current senior majoring in Computer Science at Wayne State University. This semester I undertook a six weeks project for one of the course that I enrolled in. The course, CSC 4420, is a computer operating systems course. The purpose of this course is to teach us the fundamentals of a computer’s operating system (OS). Some things learned through the first six weeks of this course are all related to system calls, library calls, kernel, file systems, ram allocation and memory allocations. After learning about how a CPU handles these events above, the final six weeks of the course was focused on creating a Secure S3FS.

***1.1 What is S3FS-FUSE?***

S3FS-FUSE is a file system that allows a user to mount an Amazon S3 Bucket to one of the user’s local directory. There from the local directory that the user chooses to mount the bucket too, now becomes a local mount point that connects the local directory to Amazon’s S3 Bucket. These buckets are used to safely secure and backup any files that is uploaded into the bucket. Once uploaded into the bucket, the user can then download the files directly from S3 or open the files from their local mounts.

1. **Project Goals**

When receiving this project, there were certain goals that were listed to us. One goal was to create a Secure S3FS system that will encrypt and decrypt any files that are put into the local mount. The second goal of this project was to develop a standalone implementation of a famous encryption algorithm that can and will function on its own.

***2.1 Secure S3FS***

The purpose of this project is to fully secure any files that the user chooses to upload into Amazon. The user will now have the privilege to have their own encryption keys and encrypt their own files other than depending on Amazons encryptions. When a user uploads a file into their local mount, it should upload straight into the S3 bucket, but when it is downloaded by either the user or anyone who has access to the bucket, the file will not viewable to anyone who does not have the correct key to decrypt the file. Figure 1 shows the process/flow chart on how S3FS should handle you files correctly.

*Fig. 1 Process Diagram*

*Simple model of how S3FS should work.*

***2.2 Standalone Implementation***

Another goal of the project is to develop your own standalone implementation of an encryption/decryption algorithm. The algorithm is chosen from a list of the built-in functions that are already predefined by OpenSSL. The standalone implementation of the encryption/decryption algorithm must be able to perform the same functions that the encryption/decryption algorithm is capable of. It must be able to encrypt any files and be able to be decrypted by using the built-in function by OpenSSL. Vice versa, your standalone should be able to do the reverse of this as well. For example, if you can encrypt using your standalone and decrypt it using the built-in, then you should be able to encrypt using the built in and decrypted using your standalone.

*Fig. 2 Standalone Implementation goal*

*Model of how the standalone should function*

1. **System**

The system that was used for the project is an Microsoft Surface Pro mode 2017. The system has 8 gigabytes of memory, 128 gigabytes of storage, and had an Intel Core 7th gen I5 chip processor. The system Dual-booted a Linux based Ubuntu.

The memory that was partitioned for the virtual box stands as follow:

* 30 gigabytes of storage memory
* 4 gigabytes of memory

The reason why 30 gigabytes was chosen for the partition is because Ubuntu’s OS takes up around 10 gigabytes already. To make sure that after installation there were still memory to work with, 30 gigabytes was the number that was decided. This would leave the virtual machine with 20 gigabytes of free storage memories to work with.

1. **Tools and Packages**

The tools and packages that are used in this project are s3fs-fuse and OpenSSL. Both of the following can be downloaded/cloned straight from their public GitHub account.

* 1. ***S3FS-Fuse***

As mentioned above, S3FS-Fuse is a free file system that is available for download via GitHub. There have been 12 different releases for S3FS-Fuse, but the most recent release is S3FS-Fuse 1.86, which I used. To clone and use S3FS-Fuse, there were some steps that was needed. The first thing that was needed to be done is to make sure that your Linux host had all the dependencies that were needed to run S3FS. To do so, you need to run the command:

sudo apt-get install automake autotools-dev fuse g++ git libcurl4-gnutls-dev libfuse-dev libssl-dev libxml2-dev make pkg-config

This command when ran in terminal will update all your libraries and download any libraries you may not already have. After doing so all that is left to be done is to clone S3FS-Fuse from their GitHub. The following command also need to be ran within terminal:

git clone https://github.com/s3fs-fuse/s3fs-fuse.git

cd s3fs-fuse

./autogen.sh

./configure

make

sudo make install

After running this, S3FS will now be cloned onto your system. Now you can use it.

* 1. ***OpenSSL***

OpenSSL is a library of many built in functions that encrypts and decrypts. When you originally downloaded your Linux system, in this case it was Ubuntu 20.04, it automatically came with it. To be sure that this project was running the most up-to-date version of OpenSSL, it was cloned from their GitHub account separately. At the time, the most recent version was OpenSSL 1.1.1.

* 1. ***Amazon S3***

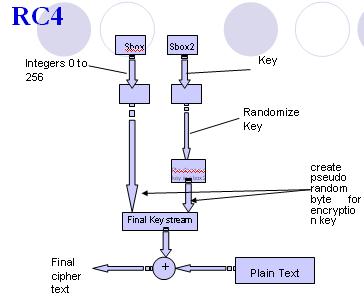
Amazon S3 is used for the storage of all downloads and uploads in this project. To use S3, first an account needs to be created with Amazon S3. You do so simply by signing up online. Once you have finished registering online, now you need to create a bucket that will be the storage of your files. When creating this bucket, you must give yourself permission to the bucket. After you granted yourself access to the bucket, it will now generate an id and password to the bucket. These two pieces of information are what’s needed to access your bucket from the local mount. The password and access id need to be saved into a .txt file and will used whenever the user is trying to mount the bucket onto the local directory.

1. **Design**

The design approach to this project was to have a secured S3FS system that encrypted files and decrypted files. Another part of the design was to have a fully functional standalone that could function the same as a built in one. To do so, what is used in the project is RC4

***5.1 RC4***

RC4 is a built-in function by OpenSSL that encrypts each file and decrypts using operations like XOR. RC4 takes in a stream of bytes, a salt vector and a key to perform the RC4 operations on the byte stream. RC4 has some built in operations in it that performs the necessary functions. One of those necessary function is setting the key. To set the key, you can call the built-in RC4\_set\_key() which takes in a parameter of RC4 key , size of the key, and a string for the key. Once the key is set, you can pass it to RC4 function. This function RC4() takes a parameter of the rc4 key, a size for the byte stream, a char\* for where the byte stream is stored and a char\* buffer for where the modified byte stream to be stored (see fig.3 for a flowchart).



*Fig. 3 RC4 Flow Chart*

*Flow chart diagram on how RC4 works*

1. **Integration**

After trying to integrate the two together there are some good and bad to the project. One good thing about it is, the standalone portion of the project is working perfectly fine. I can encrypt and decrypt using my standalone vs the built-in. The only problem is when integrating with S3FS. When uploading into S3, it is uploading encrypted, but when you download, you can’t decrypt using the built-in or the standalone. Also salting the algorithm also does not work. Actual integration of the two, S3FS and Standalone, were a problem as well.

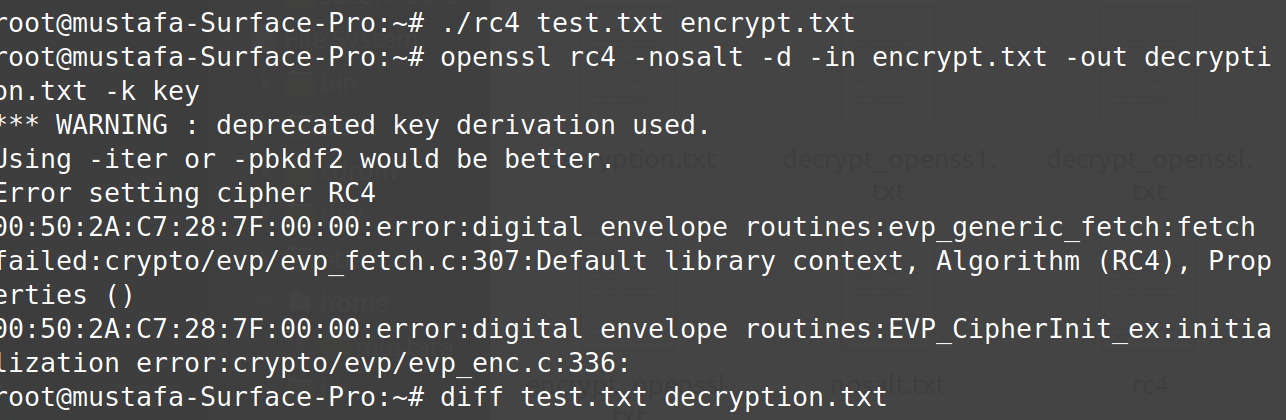
***6.1 S3FS Integration***

To integrate S3FS and an encryption/decryption algorithm, first you need to know where both encryption and decryption took place. Without knowing where they occur, you will never be able to correctly integrate the two together. You must trace the code from the main function, function by function, to see where they occur. S3FS is built with many different header files and cpp files and they all function together. By tracing the code, it will tell you what each line that calls a function comes from, and what that function does. When the location of the encryption and decryption has been located, now you will need to implement your functions to either encrypt or decrypt there.

Unfortunately, the integration in my project doesn’t function correctly. The problem is it won’t decrypt using my built-in function nor the built-in OpenSSL. The reason is probably because of the way S3FS handles the fd difference. Even though I just copied over my same function from the standalone to the S3FS, it doesn’t allow any decryptions. Before the encryption, I believe that S3FS may handle the fd a bit different than how its handled in the standalone. This could be the reason why decryption using the standalone or built-in doesn’t work because of the significant difference in the fd.

* 1. ***Standalone Integration***

The standalone integration needed to integrate both the functionality of the built-in from OpenSSL and the functionality from S3FS. For example, it must be able to encrypt using SSL built-in and decrypt using your standalone. Vice versa it needed to do the same thing when you encrypt using your standalone (see fig. 2). The point is you needed to integrate what ever OpenSSL used in their built-in functions. Only then would your standalone be able to encrypt and decrypt with OpenSSL. The difficulty with this is that you needed to know what exactly OpenSSL was doing with their keys and files. Thus, you now need to trace in OpenSSL to see exactly what it did, so you can mimic it and integrate it into your own standalone. My standalone consists of using the hash function MD5 and calling the Rc4 set\_key and the RC4 functions to mimic the SSL built-in. It functionally decrypts any encrypted files that were encrypted by my standalone with OpenSSL. Below is a screen shot of my terminal after encrypting with my standalone and then decrypting with OpenSSL.



*Fig. 4 Decryption with OpenSSL output*

*The photo shows encryption with my standalone, then decryption with OpenSSL*

* 1. ***Salt – unfinished***

By default, every operation whether it be encryption or decryption has salt on it. RC4 by itself is too weak and easily can be decrypted. This is the reason why salt is added to the encryption to strengthen the encryption, so it is not easily decrypted. Salt is a vector of randomly generated bytes added to the encryption to make things harder to decrypt.

Unfortunately, in this project I am unable to properly incorporate salt to it. When trying to use salt in my encryption I am unable to decrypt with my standalone. Also, I am unable to also unable to decrypt using the SSL decryption. The biggest problem here is since the key is randomly generated, I am unablehgk to keep both keys value the same. Thus, when I try to decrypt using my standalone, it doesn’t work because the key values of the salt vectors are different.

* 1. ***S3FS Standalone integration - unfinished***

While trying to integrate there was one issue that arose. Earlier in the class I originally demoed my project with the integration working. This was before I had the decryption with OpenSSL working correctly. When I re-demoed to TA, my integration part did not work. This was probably because my fd sizes were probably handled differently by S3FS than I expected. If I were to download a file from my S3 storage and move it into my S3FS local mount folder, this would correctly decrypt the file. The only issue was when I download a file and try to decrypt with standalone or OpenSSL.

1. **Implementation**

The plan to implement this was to first integrate S3FS with an easier algorithm and make sure that it works correctly. This first plan did a lot more than most would expect. It may be an easy algorithm, but it shows you if you are working on the correct portions of the codes or not. The algorithm was small enough for you to debug any bugs and allowed you to understand what was going on when it encrypts and decrypts. This was key in this project because it made the integration easier.

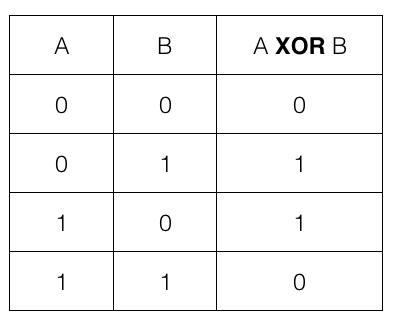
After implementing the integration of S3FS with an easier algorithm and making sure it works, the next plan was to now focus on getting the standalone portion to work correctly against the built-in. Since S3FS was going to be tested against your standalone eventually, you need to make sure that you get your standalone correct. If you got it correct, then all you need to do is copy and paste the function into your S3FS and you will be done. Since they are then going to be both the same function, you know that this will allow them to work against each other correctly.

The last step to the implementation was to implement the function from the standalone into S3FS. Once you know that your standalone is fully functional, you can be sure that when you integrate that with S3FS, it will be functional also. Thus, meaning you will have a fully secured S3FS and functional standalone function which encrypts and decrypts.

* 1. ***S3FS XOR Implementation***

As described above, the first approach was to design a small encryption/decryption algorithm that would function. Professor Xu suggested that the class come up with their own XOR algorithm which would encrypt the bytes of the file with xor ‘0’. This was the perfect starting point because it showed you how to read and write to the file descriptor (fd) and how to modify it. After learning how to modify the fd, this was already half of the XOR algorithm and it also showed you how the built-in functions were going to work as well.

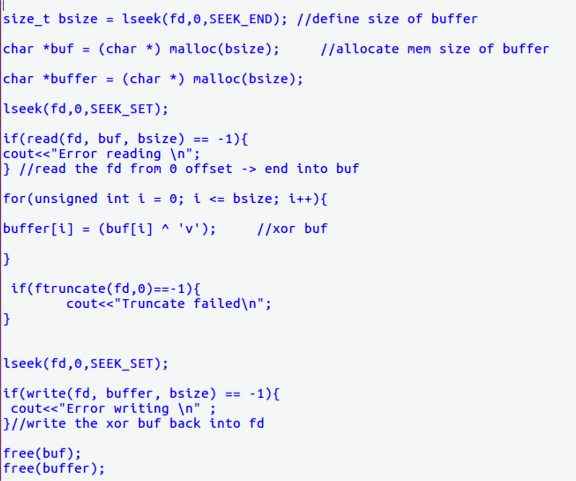
The XOR algorithm consisted of a couple of calls that was taught in class: read, write and lseek. Read call was used to read all the bytes from the fd and store them into a char\* buffer. Within this char\* buffer, this is where all your modifications/xor operations were to take place. Once the bytes were correctly read into the buffer, you can now perform xor (^) operation on them. Now the buffer stores each byte into an array, which allows you to modify they bytes easily. After the modification of the bytes inside the buffer, you need to write them back into the fd. Now remember that we already used read on the fd earlier. The read function call reads each byte within the fd into the buffer, but once it gets to the end of the fd the offset stops at the end of the fd. This means that if you try to write something else into the fd, it wouldn’t work because the fd offset is already at the end of the byte stream already. You will need to seek back to offset 0 in the fd, which is the start of the fd byte stream. Only now can you over write each bytes that’s contained inside of the fd with the modified bytes from the buffer(see fig 4 below for code). Since we are using an XOR operation to encrypt, we should be able to use the same function to decrypt as well. XOR operations works kind of in reverse. For example, if we were to perform 1 ^ 1 during encryption, the new byte would be 0. To decrypt this byte, we would take the encrypted byte 0 ^ 1 which will give us our original byte before encryption of 1 (see fig. 5).



*Fig. 5 XOR Truth table*

*Truth table for XOR operations*

Using the idea above, we are now able to encrypt and decrypt using your own built-in XOR function. From earlier traces that was performed, I found out that the areas that needed to be modified to call the XOR function to encrypt and decrypt was inside the fill Fdcache.cpp. Within this file is where it handles the cache before the actual upload/download of the file. The specific functions that are going to be worked on are Fdentity::rowflush() and Fdentity::load(). The encryption takes place inside of rowflush and decryption takes place inside of load. Since this is an XOR operation we can call the same function but put them into two different places to encrypt and decrypt. Below you can look at my own take at a standalone XOR encryption algorithm I came up with.



*Fig. 6 XOR Algorithm*

*Standalone XOR code written by me*

* 1. ***Standalone Implementation***

The standalone implementation is the implementation that can be used to encrypt and decrypt files against the built-in OpenSSL. First off you needed to know how exactly OpenSSL Rc4 algorithm worked to come up with your own standalone. When encrypting with OpenSSL, it first modifies the key with a hash algorithm. After it performs the hash on your key, it then passes the key to the Rc4 functions set\_key which sets the hashed key as the key for the file. It then passes this key to the Rc4 function RC4 which will now actually perform the encryption using the key that was set earlier.

Also included inside of the standalone are the RC4 libraries that are needed to call the built-in functions. This is not necessary, but it was done to ensure that anyone who was to use the standalone would be able to use it with out downloading OpenSSL.

After learning exactly how Rc4 works, now you will now be able to mimic it. Some calls you will need are open, read, lseek and write. Similarly, to the XOR function, you will need to read from the fd into the buffer. In the buffer is where you will make all your modifications. To mimic the way OpenSSL handles your key, you need to perform a hash function for it.

Similarly, decryption using RC4 could use the same encryption algorithm. RC4 is very similar to XOR operations, thus means that you can use the same encrypt algorithm to decrypt it.

* 1. ***S3FS Standalone Implementation***

The approach to integrating the two was to copy over my code from standalone and into S3FS. I replaced all the areas where I originally made calls to the XOR function. This included the function calls to XOR that was placed inside of rowflush and load. RC4 functions like XOR when it comes to operations so in this case we can also call the same function in the two same places where we called the XOR. The function that you copy over must have a defined key already for this to work. While in S3FS the fd is taken care of S3FS already so all you need to do is call it from anywhere within fdcache.

When copying the code over you must make sure that you have the same libraries that your standalone has for this to work. This is because if your standalone uses a openssl library, but you don’t include it in your S3FS you will get an error because the calls from your standalone won’t be defined because it can’t find the library. In my standalone, I use the library md5.h. This library includes the hashing function that hashes a given key before actually setting the key to the RC4\_set\_key(). You must also include from the openssl library rc4.h. This is the library that would allow you to call the functions set\_key and RC4 functions.

The test to make sure that the S3FS integrated with standalone works was the upload of a .jpg file. You had to first copy and paste a .jpg file from somewhere else and into your local mount. There you must check the web version of S3 to make sure that your S3FS is uploading the file and if the file is encrypted or not. The second step was to download the document and try to decrypt it with your standalone implementation. If everything goes right, you should end up with the same files because your encryption and decryption will be using the same key and same functions. Similarly, you should be able to encrypt any .jpg files with your standalone and copy the encrypted file into your local mount. This should also be able to decrypt your encrypted file if it works correctly.

1. **Future Improvements**

Even though there have been much that was accomplished in this project, there are still much more that can be better improved. First thing is I believe that without the salt portion working, the RC4 is too weak and won’t be good to use. Second thing is the efficiency of the encryption can be improved by implementing EVP calls that are from OpenSSL. Lastly, one other thing that could be improved is the actual integration part of standalone to S3FS.

* 1. ***Salting***

The salting portion of the project is a big one because its what makes the encryption so secured. Without it, the encryption is weak. If I had a little bit more time on this project, it would be dedicated incorporating salt into the algorithm. I would also like to spend more time figuring out how salt is implemented through RC4. This would help a lot because then it would be easier to mimic in your standalone.

* 1. ***EVP Calls***

Upon completing the assignment, I stumbled on something called EVP calls. I did some research on it on the OpenSSL website and found out that they could have made life so much easier. EVP calls could have done the whole project but made it more efficient and easier to operate. They perform certain duties that it would take me 5-10 lines of code to do. The built in EVP calls does not need to be freed like how a char\* buffer would need to be freed at the end of each functions. They also provide a more secure hash function and encryptions and decryptions.

* 1. ***Integrations***

If there were more time for me in this project, I would also spend on getting my integration portion correct. As mentioned above, everything works fine with no salt. The only thing that doesn’t work is decrypting downloaded files from my S3. This is a small bug that will be an easy fix, but the issue is the time to debug it. Once the problem has been located from debugging, the fix is simple, I believe. The only issue is debugging. When I diff the two pictures, the downloaded encrypted picture and I decrypt with OpenSSL, there is a difference between the two.

1. ***Summary***

On April 20, it was the end to this six weeks project that was assigned to the class. Initially before starting the project, the class knew that it was going to be a hard project. There were nights when sleep was not an option because the weekly project was not finished. After lots of hours of debugging, tracing, reading documentations, and trial and error, the project is about 80 percent completed. As highlighted above, there were many things that was not able to be finished, but there are some things that’s notable. Also, the project taught me a lot of very useful things that will be only needed for this class but needed for any Computer Science major in their future. The project ended up being very fun even though it was somewhat challenging.

* 1. ***Accomplishments***

Some accomplishments that came from this project was being able to build a functional standalone XOR encryption/decryption algorithm, being able to recreate a standalone implementation of RC4 with no salt and becoming more efficient with using terminal on a Linux OS.

By creating my own algorithm, it forced me to trace the code of S3FS to find out how it’s exactly uploading and downloading files. This gave me lots of insight on how a file is being uploaded either by S3FS or any other sources. As it was said in class by Professor Xu, creating your own sandbox/algorithm is very difficult. Even though all that was created by me was just an XOR, it’s still something I am proud of doing because I built it. Not only was it just built, it was functional in the sense that it did encrypt and decrypt any files that are put into my local mount. Just from asking around in the class, I know that most people had difficult times with creating their own XOR algorithm that would function correctly, so in a sense I am proud that I got mines to work.

Creating a standalone is also a good accomplishment that is good to be proud of. Again, from asking around the class, most people also do not have this feature completed as well. This part was a difficult one because most people would just try to call the functions set\_key then RC4 and expect it to work but this is not the case. This forced me to dig through OpenSSL’s source codes and find out what exactly is being done to the key to mimic it. I believe that this has further enhanced my debugging and tracing skills of any large codes and for that reason it is an accomplishment I am proud to brag about.

***9.2 What I learned***

Something else I am proud of is becoming more efficient with terminal in Linux. At the start of the course I can say that I had no idea what was being taught to me because of never using terminal before. As the course progressed, more and more calls were learned but there really weren’t any thing to test these commands on other than practicing them on your own. This project was a very good test subject to use these commands on. Sure, the project could have been also completed using GUI, but we all know that it is a very slow and inefficient way to work. GUI is designed mainly for end-users and has bad efficiency. If the class were to only use the GUI to demonstrate their demo and use the GUI to complete the project, it would take very long to do so. Some commands that were used a lot in this project by me were cp, rm, df -h, umount, make, diff, and ll. These are all very basic commands in the real world, but we all must start from somewhere and just getting used to these basic commands really helped make this project easier for me. Instead of having to use GUI to copy files from one folder to my local mount, or vice versa, I could do it in two commands in terminal and check to see if it really copied the right files or not and even renaming the copied file. Another command/functionality in terminal that I learned more about was permissions to files. Again, repeating what’s stated above, if I were to manually use GUI to change the permission of a file, it would be much more inefficient than using terminal chmod to do so. As Professor Xu said in class, “If you can’t do these basic commands, even if you pass the course with a good grade, you haven’t really passed this course”. This is actually very true if you were to think about it because if you just pass this course with an A for example but can’t even do a simple cp command and just uses GUI for everything, then you haven’t learned anything useful that can be used in the real world and job interviews are going to be very difficult for you.

The most important thing that I learned from this project/from this class is to never give up. Anything is possible if you just put in the time and hard work. When the project was given to the class, no one went over what S3FS files does, or which files are the ones to be modified. It was lots of hard work and time to debug and trace through all the different source files and main files to find out where exactly the encryption and upload takes place. Also, something that took time to do was learning how files were being uploaded. You really had to understand what is going on to know how you can modify the fd to encrypt it. There were times when it felt that giving up was the best option but looking back now and I’m glad I pushed through it all. These aren’t even skills that you need, they are just your will that you need. When things get harder and you feel like there’s nothing you can do, you just must push through it.

1. **Closing**

To close this, here is a summary of everything included in this report. The project was designed to modify your S3FS to become secure by encrypting files that you upload into S3. To do so, you must create your own standalone implementation of OpenSSL RC4 to encrypt and decrypt files to and from S3. After completing your standalone and you have checked it against the built-in OpenSSL version, now you integrate it into your S3FS. Upon doing so you must check that your S3FS can now upload and download encrypted files and decrypt them as well. Also, you should be able to decrypt them against your standalone and OpenSSL as well. If all works, then you will know that you have a fully secure S3FS.

The project has been a very fun project and very worth it. This project will stay with me forever, and it is a project that I will be very proud of to show off to other peoples and even to future employers at interviews. Even though the project ended, it hasn’t really ended for me. There are still many flaws with my program that I believe I can fix and I will fix it.