NANYANG TECHNOLOGICAL UNIVERSITY

HYBRID CLOUD STORAGE AGGREGATOR FOR PRIVATE USE SCE15-0036

Supervisor: A/P Anwitaman Datta
Examiner: A/P Tan Kheng Leong

FYP Final Report

By

Shun Hanli Hanley
U1321587E

SCHOOL OF COMPUTER ENGINEERING 2015/2016

ABSTRACT

In this report, we document the implementation of the protocol that allows end-users to protect data that are sent to remote servers using two factors – knowledge (passwords) and possession (a time-based one-time password generation for authentication). The implementation does not require any trusted third-party and also supports invoking a new possession factor in the event that the older possession factor is being compromised provided that the legitimate owner still has a copy of the possession factor. The approach protects the outsourced data from the storage servers themselves by encrypting and dispersing the information parts across multiple servers. The basic protocol is also extended to demonstrate how collaboration can be supported even when the stored content is encrypted, and where each collaborator is still restrained from accessing the data through a multi-factor access mechanism[4].

The scope of this project has strictly been implementing the protocol and not in the design of the basic protocol or its extensions.

[4] Ertem Esiner, Anwitaman Datta. (2015). Layered Security for Storage at the Edge: On Decentralized Multi-factor Access Control. Nanyang Technological University

ACKNOWLEDGEMENT

I would like to express gratitude to the Project Supervisor A/P Anwitaman Datta, for the opportunity to provide input on an important topic that has become popular in the recent years of rapid technological advances.

I would also like to express appreciation to PhD Candidate Ertem Esiner for the constant guidance and assistance. The insightful feedback and support has benefited me very much.

FIGURES & TABLES

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 MOTIVATION

This work serves as a proof-of-concept and is a realization for a decentralized multi-factor access control as layered security for storage at the edge.

Decentralization is the process of dispersing or redistributing functionalities away from a central authority. So instead of passing data and information through a central hub, the central hub is done away and peers send data back and forth directly to each other. It is needed because it is faster, cheaper and it makes resources more accessible. However, a decentralized network is not easy to build because it is difficult to plan, design and manage. Having Multi-Factored Access on the decentralized network makes matter worse. Security becomes a problem such that files, when being stored on a decentralized server such as a cloud, are compromised when the servers are compromised.

In many of the distributed data storage settings, there has been no work done with respect to such a protocol. As such, the need for security in this area motivated us to take a step further to implement our concept and prove that such a protocol is required and works.

1.2 PROJECT OBJECTIVE & SCOPE

The objective of this project is to create a workable implementation of the protocol as a proof-of-concept. In order to fulfill this objective, several open-sourced libraries in the internet will be used. However, the project is not limited to the completion of the implementation.

The purpose of this report is to document the entire process of the proof-of-concept, the set-up and the final product of the implementation. The performance of such an implementation will also be measured and subsequent improvements or future work will be discussed towards the end of the report.

The project has been in progress since the beginning of AY2015/16 Semester 1 and will conclude at the end of AY2015/16 Semester 2.

1.3 WHY IS THIS NEEDED?

In this report, we implement a workable solution for the proposed protocol to prove that layered security can be achieved for decentralization, specifical, cloud computing. We also will show that IT security adds value to decentralization by having this working implementation of layered security.

A layered security solution for cloud storage does not improve the trustworthiness of all cloud storage providers. Instead, the solution gives credibility to the information and protects it by denying access to anybody who does not have the knowledge and possession factor needed. It can be argued that even if the cloud storage provider becomes compromised by an adversary, the confidentiality and integrity of information stored on these servers remain intact.

This implementation is useful because the backend processes are deliberately constructed to be transparent to the user. Whenever a user uploads a file to the cloud storage, the protocol allows for the encoding and encrypting of that file simultaneously.

However, this project had been a difficult journey and the objectives were hard to achieve because of the unavailability of resources required to complete the proof-of-concept. Nonetheless, careful and precise use of open-source libraries has enabled the POC to be a successful one. Many other works and articles have discussed the importance of having layered security[5] and layered security on cloud computing. An article on USA today[6] highlighted the fact that cloud providers must adopt a defense-in-depth strategy for data security. The consequences of Decentralized Security[7] were also discussed. Unfortunately, all of the past works have been inadequate in proving that having layered security for decentralization is feasible since they had not proposed a protocol and a working proof-of-concept. For example, in a paper published by the University of Putra Malaysia titled 'Formulating a Security Layer of Cloud Data Storage Framework Based on Multi Agent System Architecture'[8], it discussed methods security issues relating to cloud and proposes a security framework but did not elaborate on the actual implementation of their proposal.

Finally, to reinforce on what this report is about, we delved deep into a protocol which provides a solution for layered security for cloud storage. Using this protocol, we created a working proof-of-concept and implemented it on two Western Digital Cloud Storage Servers and subsequently analyzed the performance of the procedure.

---

[5] Jerry Shenk. SANS Institute InfoSec Reading Room. 'Layered Security: Why It Works'. (2013)
[6] Rajiv Gupta. 'Special for CyberTruth'. (2014)
<http://www.usatoday.com/story/cybertruth/2013/11/25/why-cloud-security-requires-multiple-layers/3683171/>
[7] Douglas Thain, Christopher Moretti, Paul Madrid, Philip Snowberger, and Jeffrey Hemmes. Department of Computer Science and Engineering, University of Notre Dame. 'The Consequences of Decentralized Security in a Cooperative Storage System'. <http://www3.nd.edu/~dthain/papers/cons-sisw05.pdf>
[8] Amir Mohamed Talib, Rodziah Atan, Rusli Abdullah and Masrah Azrifah Azmi Murad. Faculty of Computer Science & IT, University Putra Malaysia. (2010) 'Formulating a Security Layer of Cloud Data Storage Framework Based on Multi Agent System Architecture'.

2.0 LITERATURE REVIEW

As technology advances from its early days, computing paradigm has been in flux, continuously alternating between centralization and decentralization. Miniaturized hardware has resulted in partial decentralization and aggrandize cloud computing and service.

Data storage is an essential function along with cloud services and computation. Security and reliability is essential for edge[6] or fog computing[7] to become the new computing solution. The focus of this work is on security and how layering it (Multi-factor access control) on a decentralized set-up can provide enough desirable trust and still maintain a low level of overhead. By augmenting the knowledge factor with an orthogonal possession factor, we go further than the normal and widely used 'user controlled encryption'.

There was a recent work on Decentralized Multi-Factor Access Control[1]. In this paper, a protocol was proposed to allow end-users to protect data shipped to remote servers. This protocol does not require any trusted third part and relies on two factors, namely knowledge (passwords) and possession (one time password generated for authentication) factor. The protocol also supports recreation of a new possession factor should the original possession be compromised. This is provided that the original owner still has a copy of the possession factor. Similar to other recent works, this approach secures outsourced data from the storage servers themselves. The data to be stored will be encrypted and dispersed across multiple servers. On top of the basic protocol, an extension of the protocol would allow for collaboration among two parties to read and write to an encrypted stored content.

These techniques of MFA control described by the various papers achieve layered security on external storage resources provided by untrusted entities.

[6] CISCO. Fog computing. https://techradar.cisco.com/trends/Fog-Computing
[7] P.G. Lopez, A.Montresor, D.Epema, A. Datta, T.Higashino, A.Iamnitchi, M. Barcellos, P.Felber, and E.Riviere. Edge-centric computing: Vision and challenges, 2015.

3.0 THE PROTOCOL

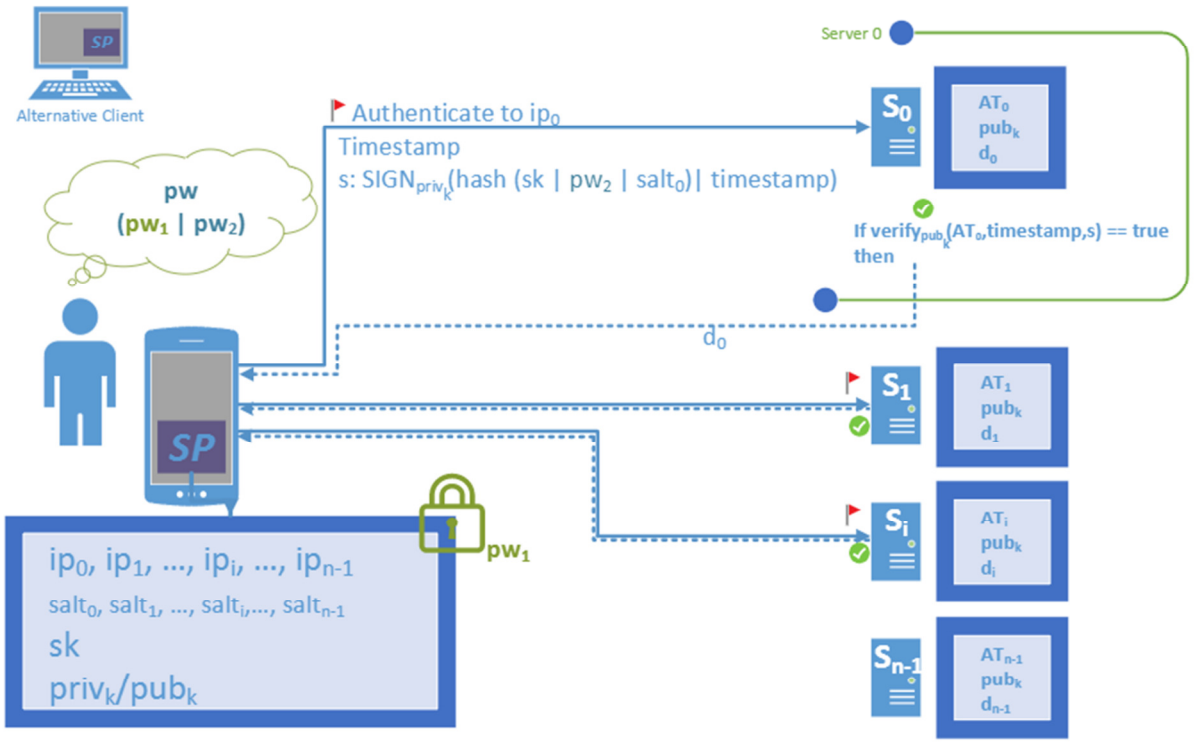


Figure 3a: The Protocol

A client will need to input a password. This password will be split into 2, pw1 and pw2 (First half and Second half), and is the knowledge factor. Subsequently, the secret package SP will be initialized as follows:

$$SP = \begin{cases} \text{ips: } ip_0, ip_1, \ldots, ip_i, \ldots, ip_{n-1} \\ \text{salts: } salt_0, salt_1, \ldots, salt_i, \ldots, salt_{n-1} \\ \text{secret token: } sk \\ \text{private/public key: } priv_k \,/\, pub_k \end{cases}$$

Figure 3b: Secret Package

The possession of SP (Figure 3b) acts similar to a token and is the possession factor. It is portable and can be used across different devices. This is very similar to the use of a secret pin for hardware tokens. The SP will be encrypted with pw1 and stored.

$$\mathbf{AT_i} = \boldsymbol{hash}(\boldsymbol{sk}|\boldsymbol{pw_2}|\boldsymbol{salt_i})$$

Figure 3c: Authentication Token

Using the secret token sk and pw2, they are appended with a salt value and hashed (Figure 3c).

When a user wants to access the original data object, the password pw = [pw1|pw2] must be provided. The correct password will decrypt and return the original SP.

$$\text{Authentication package} = \begin{cases} timestamp \\ SIGN_{priv_k}(AT_i' | timestamp) \end{cases}$$

Figure 3d: Authentication Package

Finally, an authentication package (Figure 3d) will consist of a timestamp, which is determined by the time of the current system, and the private key signing of the authentication token with the timestamp.

To authenticate, the client sends a timestamp and the authentication token to the server for verification. Using the public key stored on the server, the client is considered genuine and authenticated only when the verification of the AT and timestamp is successful.

A client may update the possession factor (SP) after successfully authenticating with the server. To update the authentication package, a new authentication package with its corresponding public key are uploaded and replaced with the existing.

4.0 SETTING UP

4.1 PRE-REQUISITES

4.1.1 DEVELOPMENT

1) PuTTY
2) Crimson Editor (Own Preference)
3) Gcc and arm-linux-gnueabi-gcc (C compiler for ARM architecture)
4) Web browser

4.1.2 SERVER(S)

1) Western Digital MyCloud Storage with SSH enabled
2) Crypto library for arm-gcc (libcrypto.so)

4.1.3 CLIENT

1) Ubuntu-Wily (Preferred) and above
2) Crypto (OpenSSL) library
3) Jerasure library
4) Gf_complete library
5) GTK+-2.0 library
6) NTU-VPN if connecting from outside of NTU network

4.2 SERVER

The Western Digital MyCloud runs on a special version of Debian. It is 32-bit and has armv7L architecture. It has been modified such that it does not allow any packages or software to be installed on it. It does not have GCC. As such, we will have to compile it locally as ARM before uploading to the WDMyCloud device for testing.

Since the server will have to perform verification of an authentication token (Signing), we require the OpenSSL library to be available on the server.

4.2.1 INSTRUCTIONS

- Set up Cloud server for development
    - On a browser, login to 155.69.144.222 and 155.69.144.223 and enable SSH
    - Scroll the tabs and go to Shares. Click on icon Add Share at the bottom left to create a new share folder called 'server'
    - Scroll the tabs and go to Apps. Click on Web File Viewer and upload 'server_arm.exe' to the 'server' folder
- Prepare environment to execute program
    - SSH into the server as user root using PuTTY
    - Execute 'cd /nfs/server'. If folder does not exist, execute 'mkdir /nfs/server'.
    - Upload libcrypto.so to server
    - Execute 'cat /etc/ld.so.conf' to find the shared library path
    - Execute 'cp libcrypto.so <path to shared library>'
    - Execute 'ldconfig' to update the cache

4.3 CLIENT

Set up a Linux environment that is running on Ubuntu, version 15.10 (Wily Werewolf) and above. The implementation requires the use of 4 open-source libraries namely, OpenSSL crypto, Jerasure, gf_complete and GTK+-2.0. If outside of NTU network, NTU-VPN is required as well.

4.3.1 INSTRUCTIONS

- Set up OpenSSL crypto (Usually already available)
  - Execute 'sudo apt-get install libssl-dev'
- Install GF-complete
  - Execute 'sudo apt-get install libc6 libgf-complete1 gf-complete-tools gf-complete-dev'
- Build Jerasure from tarball distribution
  - Download Jerasure library from http://www.kaymgee.com/Kevin_Greenan/Software_files/jerasure.tar.gz
  - Extract to current working directory and execute 'cd jerasure'
  - Execute './configure'
  - Execute 'make'
  - Execute 'sudo make install'
- Install GTK+-2.0 for the GUI (Required for GUI)
  - Execute 'sudo apt-get install gtk+-2.0' (Installation will take awhile)
- Setting up gcc and arm-linux-gnueabi-gcc for Development
  - Execute 'sudo apt-get install build-essential' (If you did not modify /etc/apt/sources.list)
  - Execute 'sudo apt-get install gcc-arm-linux-gnueabi'
  - Execute 'sudo apt-get install putty'

5.0 IMPLEMENTATION

5.1 STEP-BY-STEP IMPLEMENTATION

The implementation can be categorized into different stages as shown in Figure 5a. The final product will be a combination of all the different stages put together, executed through a Graphical User Interface.

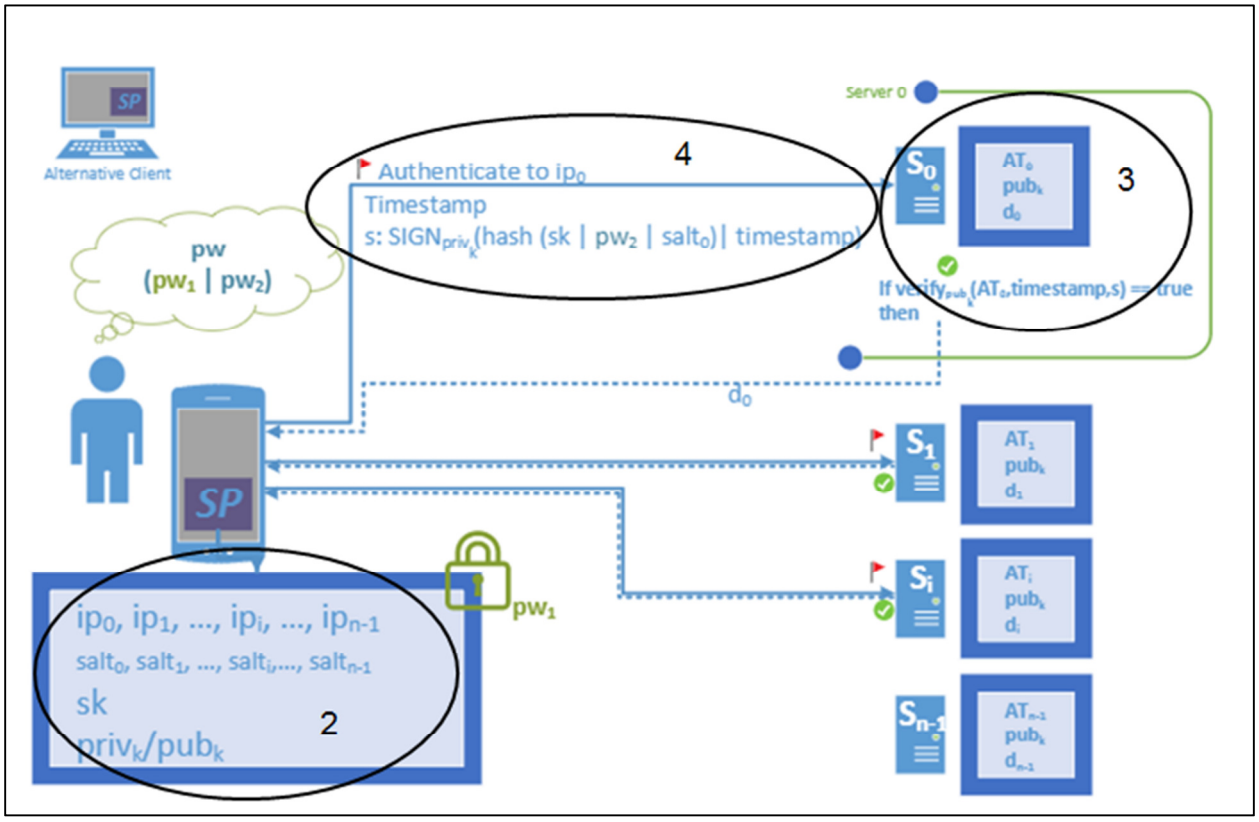


Figure 5a: Implementation Groupings

The implementation stages are as follows -

1) Encoding and encrypting of file
2) Creation of secret package
3) Signing of authentication token
4) Authentication and socket file transfer
5) Updating possession factor
6) Upload & Download file
7) Decrypt & Decode file

5.1.1 STAGE (1): ENCODING AND ENCRYPTING OF FILE

In this implementation, we adopt the Classic Reed-solomon codes. The open-source Jerasure library will be used. This implemention allows us to split the original file into k-number of parts and encode them into m-number of encoded parts.

In the beginning, a file will be encoded - enc(k, m). The m-number of encoded parts will subsequently be separated and uploaded into multiple servers. Reed-solomon codes works in such a way that if any part of the encoded file is corrupted or missing, the original data may still be assembled back to its original[8].

The successful encoding of the file will generate k-number of parts of the original file, m-number of encoded parts of the original parts and a meta-file that contains information of the erasure coding. This meta-file will be essential for the file to be assembled to its original file.

Subsequently, all the files generated from erasure coding will be encrypted (Metafile will be encrypted as well) with pw1 and will only be retrieved if it is to be decrypted with the same pw1.

5.1.2 STAGE (2): CREATION OF SECRET PACKAGE

In Stage 2, a pair of public key and private key is generated and stored as public.pem and private.pem respectively. Along with the key pair, a secret key, which is a string of 256 characters, consisting of alphanumeric and special characters, is also randomized.

With reference to Figure 3b, the IP addresses of the servers, their respective salt values, the secret key and the public and private key will be serialized into a file called secretpackage.data.

At this point, the knowledge and possession factors are created, encrypted and stored in the secret package on the client side. The secret package also contains the IP addresses of all the cloud servers with its respective salt.

5.1.3 STAGE (3): SIGNING OF AUTHENTICATION TOKEN

Referring to Figure 3d, an authentication package is created by hashing the secret key, pw2, salt and timestamp altogether. It is then signed using the private key (private.pem), resulting in an Authentication Token (AT). The timestamp is stored as a separate file for authentication purpose in the authentication stage. Prior to storing the AT(s) to the cloud servers, the AT(s) are verified locally to ensure that the signing has been done properly.

5.1.4 STAGE (4): AUTHENTICATION AND SOCKET FILE TRANSFER

[8] James S. Plank, Jianqiang Luo, Catherine D. Schuman, Lihao Xu, Zooko Wilcox-O'Hearn. A Performance Evaluation and Examination of Open-Source Erasure Coding Library For Storage. https://www.usenix.org/legacy/event/fast09/tech/full_papers/plank/plank_html/

To start the authentication process, an AT and timestamp must be sent to the server. The client is considered authenticated only when the verification process on the server returns a positive result. The verification process is done by verifying the AT with the client's public key.

The socket file transfer has been implemented to allow the download and upload between client and server. Using this, the respective ATs and encrypted data parts, along with the public key, are uploaded and stored on each server. This is also essential for the authentication process in the next stage. Figure 5b shows the result of stages 1 and 2.

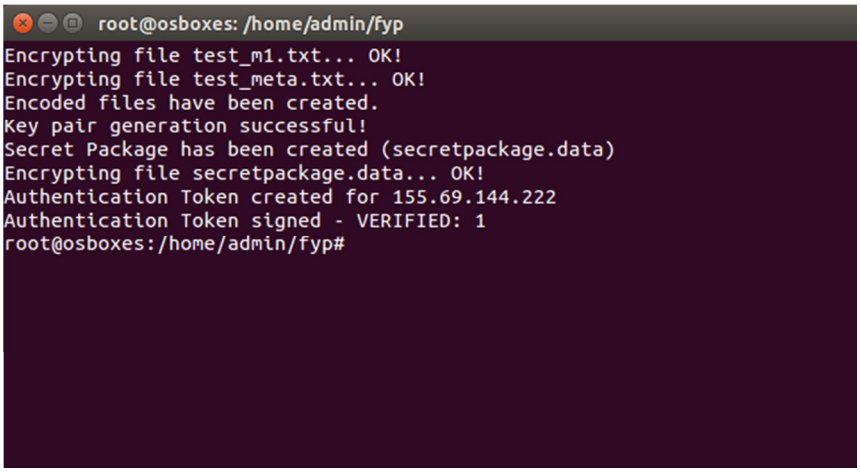


Figure 5b: Secret Package and Verified Authentication Token

The socket file transfer will also be used as the medium for the (6)Upload and (7)Download of files.

5.1.5 STAGE (5): UPDATING POSSESSION FACTOR

Upon successful authentication to the server, the client is able to update the possession factor by uploading a new Authentication Token (AT) and its corresponding public key. The new AT and public key will replace the old AT and public.pem that is residing on the server.

5.1.6 STAGE (6): UPLOAD & DOWNLOAD FILES

The file socket transfer allows for bidirectional multiple files transfer. Client may select more than one file to upload to a server. A client may select download files as well, of which the server will receive the download request and respond with a list of files that can be downloaded by the client. Client subsequently selects the file(s) and server will send them.

5.1.7 STAGE (7): DECRYPT AND DECODE FILES

When a client has downloaded all the different parts of the encoded and encrypted files, the files will be decrypted with the same pw1. Without the password, the client will not be able to decrypt the files.

Upon successful decryption of the files and the metafile (assumed to be available), the client can then decode the files by selecting the appropriate metafile. The decoded file will be labeled as '<original filename>_decoded'.

5.2 FINAL PRODUCT

Figure 5c shows the Graphical User Interface (GUI) of the final product. Figure 5d show the stages of implementation labeled accordingly to each button. The purpose of a GUI is so that the different stages of the implementations are combined into one application. It also provides a user-friendly platform for the client to perform the necessary actions.

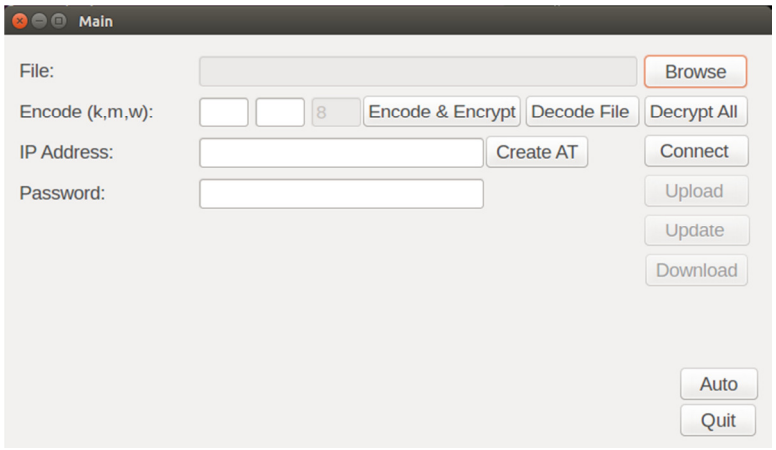


Figure 5c: Final Product GUI

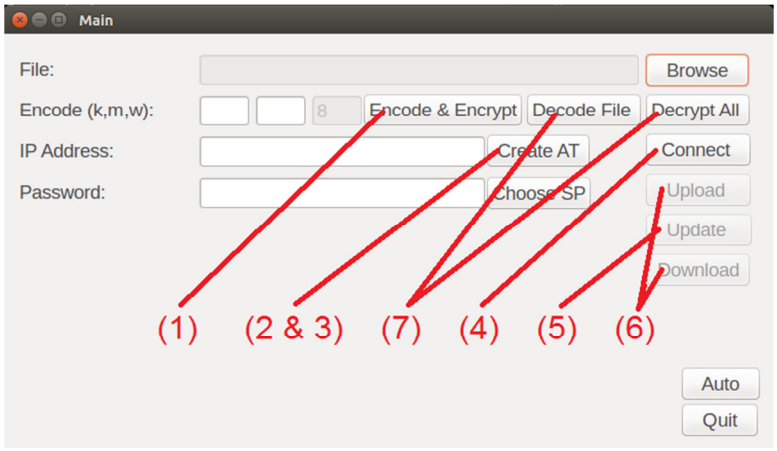


Figure 5d: Final Product GUI Labeled

5.2.1 FUNCTIONALITIES

Each button labeled in Figure 5d will perform its respective function with the proper input. These buttons are independent of each other. The file browser has been included so that the client does not require typing the path of the file.

The 'Auto' button placed at the bottom right of the GUI, above the Quit button, automates the process from Stage 1 to Stage 6 of the implementation. Several screenshots have been included in Appendix A to illustrate the program flow of the automation.

Stage 7 will be performed manually. Upon clicking on 'Decrypt All', the application will decrypt all files that ends with '.encrypted' with the first half of the given password. Subsequently, 'Decode File' button prompts for a meta-file. Selecting the corresponding metafile will load all the existing parts of the encoded file and decode them to ultimate re-create the original file.

5.3 PERFORMANCE

Performance is always been a topic for discussion when it comes to strengthening information security. The strength of information security and the overhead needed to maintain that level of security are proportionate to each other. In short, the stronger the encryption, the slower computing will become.

In this section, we determine the feasibility of implementing this protocol on decentralization, using a few tests. We measure the time taken to –

1) Encode a file of varying sizes
2) Encode and encrypt a file of varying sizes

We also note some factors that will be kept constant in this measurement –

1) Password
2) Empty folder (No overwriting and file checks)
3) Contents of files

File Size (bytes)	k,m	Time(ms)	k,m	Time(ms)
10	2,3	3.251	2,5	3.735
1000 (1KB)	2,3	4.871	2,5	5.01
100000 (100KB)	2,3	7.035	2,5	7.817
10000000 (10MB)	2,3	100.963	2,5	149.871
100000000 (100MB)	2,3	962.804	2,5	1517.73
10000000000 (1GB)	2,3	8371.83	2,5	-

Table 5a: Encoding Performance

File Size (bytes)	k,m	Time(ms)	k,m	Time(ms)
10	2,3	3.586	2,5	4.215
1000 (1KB)	2,3	5.214	2,5	5.572
100000 (100KB)	2,3	8.895	2,5	9.788
10000000 (10MB)	2,3	248.408	2,5	407.862
100000000 (100MB)	2,3	1848.683	2,5	2950.337
10000000000 (1GB)	2,3	-	2,5	-

Table 5b: Encoding and Encrypting Performance

Table 5a shows the time taken to encode files of varying sizes while Table 5b shows the measurement of both encoding and encrypting of the files. The sample files increase by a 100 times for files size of less than 10MB. The file test for 100MB is included because the program was unable to decode the file of size 1GB. This is due to the limited space of the client's machine.

Subsequently, the table for the encryption time can be derived from the above tables.

File Size (bytes)	Encoded files + Metafile	Time(ms)	Encoded files + Metafile	Time(ms)
10	3 + 1	0.335	5 + 1	0.48
1000 (1KB)	3 + 1	0.443	5 + 1	0.562
100000 (100KB)	3 + 1	1.86	5 + 1	1.971
10000000 (10MB)	3 + 1	147.445	5 + 1	257.991
100000000 (100MB)	3 + 1	885.879	5 + 1	1432.607
10000000000 (1GB)	3 + 1	-	5 + 1	-

Table 5c: Encrypting Performance

From the above tables, we plot a line graph of the file size against the time taken.

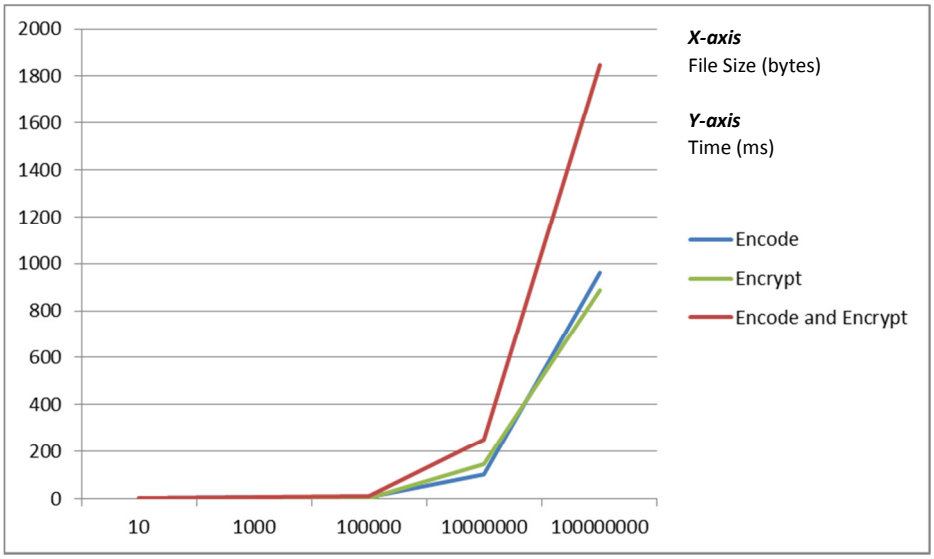


Figure 5e: Line Graph of Encoding(2,3)

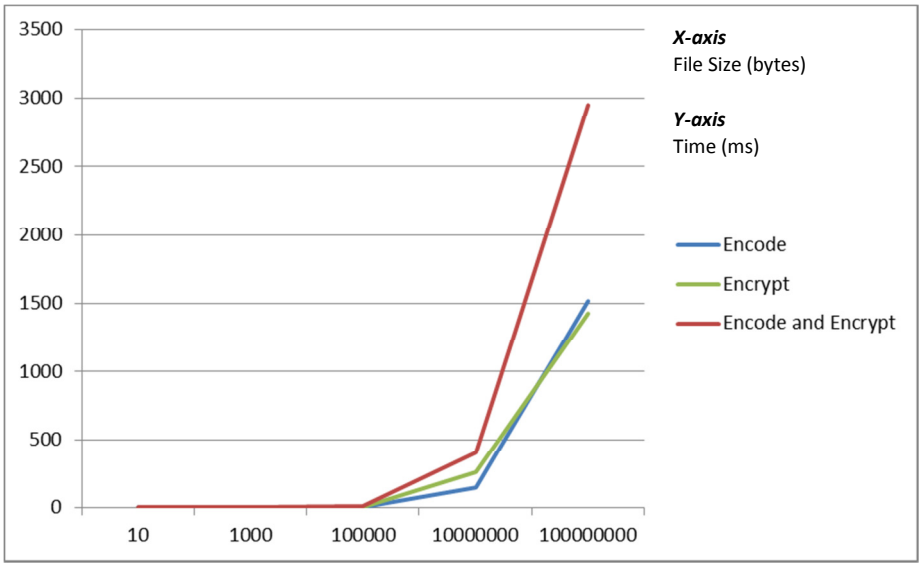


Figure 5f: Line Graph of Encoding(2,5)

For both encoding options of (2,3) and (2,5), the relationship between the file size and the time taken is similar for all 3 operations. All of the operations increase exponentially with file size and time taken. As the file size increases over 100MB, the time taken increases sharply. This could be because the operations require the machine to allocate huge amount of memory space, further slowing down the computing process.

Encoding files require less time as compared to encrypting them for files below 100MB. This shows that encrypting files of less than 100MB is less efficient than encoding them. However, encrypting the files becomes more efficient than encoding them when the files are more than 100MB. This is shown from Figure 5e and 5f when the lines intersect with each other.

6.0 CHALLENGES AND SOLUTIONS

SERVER PROBLEMS AND SOLUTIONS

Western Digital MyCloud is running on a special kind of Debian OS. As seen from figure 5g, standard commands such as apt-get, aptitude and make, had been removed from the servers. I tried manually downloading the packages and uploading them into the cloud server for extraction and execution. Still, the packages could not be installed because the servers run on ARM architecture as well. This means that standard Executables or binary files will not be able to run on it.

As such, performing any installation of packages and software, essential and mandatory in the development on the servers was not possible.

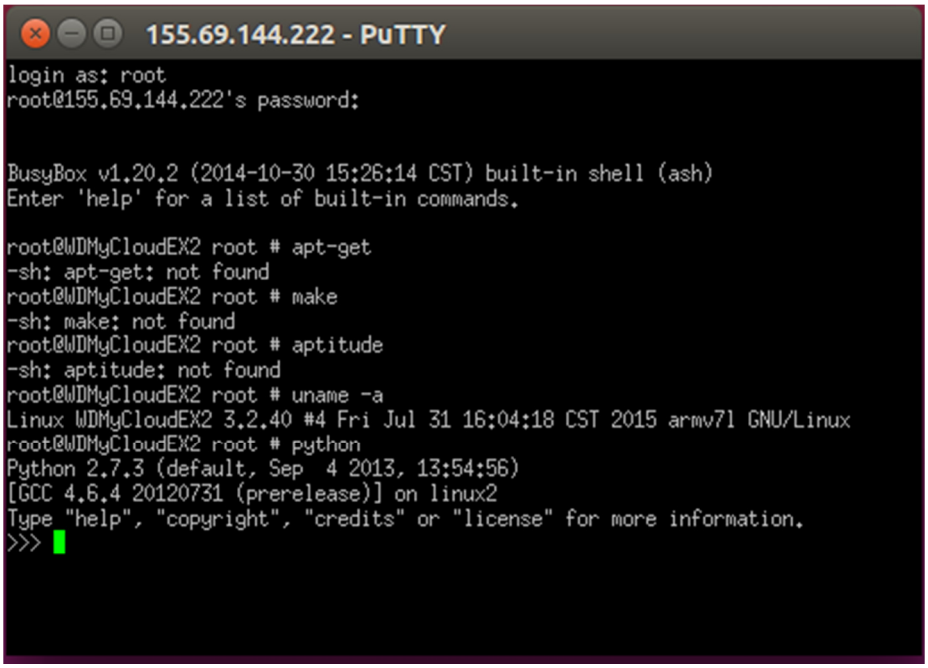


Figure 5g: WD MyCloud Command Line

I began looking for pre-installed binaries and libraries to see if there was any way I could set up the servers to allow for some form of development. I realized I could run python commands. Now I had an option of beginning development using Python. Although I am familiar with Python language, I realized there might be issues implementing OpenSSL, reason being I was more familiar with other languages like Java or C.

I tried looked for JDK on the server and unfortunately, it could not be found on the MyCloud server. I end up writing a C program on my Linux Ubuntu. I uploaded it to the server and was disappointed when it could not run. It gave the same error that said 'Unable to execute binary file.'. Fortunately, I have learnt that ARM architecture works differently as compared to non-ARM operating systems.

I went back to my local machine, looked for an ARM compiler for gcc (arm-linux-gnueabi-gcc) and compiled a 'Hello World' program with it. Going through the web interface of the cloud servers again, I uploaded the C program and ran it on the server. Voila! It worked and I saw 'Hello World' on the output. Because all the libraries I require to implement this proof-of-concept are available open-source in the internet, it was decided it would be best to perform development using C programming.

So the tool needed here was a C compiler for ARM. I need to ensure that 'Hello World' was not the only program that can be executed on the server. I wrote a C program that listens for a number from my development machine. I ran and the server could successfully receive the number and close the socket properly. I went further to perform some arithmetic on the number received and reply the sender with the results. It worked.

After getting ensuring that a C program could perform the basic functionalities on the server, there was another major problem. The crypto (OpenSSL) library for the ARM compiler was not available. Without the OpenSSL library, I will not be able to implement the verification and signing of the authentication package. It was not as simple as installing the OpenSSL library for the standard C compiler (GCC) with the command 'sudo apt-get install libssl-dev'. There were two solutions to solve this; either 1) re-create the verification process from scratch instead of using the library or 2) re-install the crypto library and build a link for arm-linux-gnueabi-gcc, and then upload the library onto the server so that the program can run. After attempting option 1, I encountered too many complications when coding the needed OpenSSL libraries from scratch. Fortunately, optn 2 worked, sparing me hundreds of hours of programming.

DEVELOPMENT PROBLEMS AND SOLUTIONS

I began development work after solving the problems of the server. Part of the implementation was to perform erasure coding on the files. I was recommended Jerasure library. While working with Jerasure library, I found that a library package (gf_complete) used by the Jerasure library was missing. Simply installing the package by 'apt-get gf-complete-tools' did not get the library to work. Unfortunately, the gf-complete-tools was not available on the version of Linux Ubuntu-Precise I was using. It was not available in the Ubuntu Package centre.

The workaround was to automatically apply an update from the OS version manager. I tried to update to the latest version Ubuntu-Wily, but failed because of an unknown error from the VirtualBox VM. Finally, the problem was solved when I downloaded a new Ubuntu-Wily image from the internet, created a new boot on the VirtualBox VM and installed all the necessary packages. It was a time-consuming method considering the new version is 9GB large. It took a few hours but it solved the problem eventually. I later had to transfer all the saved data over from the old image. It was time-consuming but it worked after many configurations.

Among other minor problems, there were improper linkage of libraries for use with the C compiler, file input and output errors and limitations of the C programming language.

These issues were solved by going to Google. I posted several questions on www.stackoverflow.com to seek help from people who had encountered similar problems in the past. The responses were of great help to me. I managed to solve many of the problems with the help of searching for known alternatives and solutions on Google.

However, the implementation of the Reed-Solomon erasure coding using the Jerasure library was tough and tedious. There was very little support online. Painstakingly, I studied studied and carefully traced thousands of lines of codes. After multiple trial-and-errors, the intended results were achieved.

The next challenge was combining all the different stages of implementations together into a final product. Because the stages were implemented independently from each other, there were many conflicting variable names, buffer overflow and inefficiency problems. As the size of strings in C program is fixed and cannot be changed, there were problems copying strings into the buffer for transferring through the socket. The solution was to use pointers instead of repeatedly copying them into new variables. I also learnt to use safe functions that limit the input, for example, using strncpy instead of strcpy. This restricted the number of characters copied and prevented the buffer from overflowing.

7.0 CONCLUSION & FUTURE WORK

7.1 CONCLUSION

Implementation of layered security for storage at the edge is feasible for home or small business use when information and data are relatively small; <100MB. When files are larger than 100MB, average computers and laptops will not be able to handle the processing. This is not taking into account the time required for the files to be sent across the network. Nonetheless, multi-factor access control can be a good way to ensure confidentiality on a decentralized storage.

Performing development on a Linux machine was a first for me. Needless to say, when the MyCloud servers ran on a special kind of Linux Debian OS, things got even more challenging.

Still, the project progressed in the right direction and on schedule. There was ample time to implement and accomplish the tasks required. Although the project was stalled for a period of time in the first half of the year due to school work and other commitments, the project caught up during the break in December.

However, there was certainly potential for further exploration. Better time management would have allowed the implementation to be completed and further development and innovation to be done and incorporated, leading to a more fruitful learning journey.

7.2 FUTURE WORK

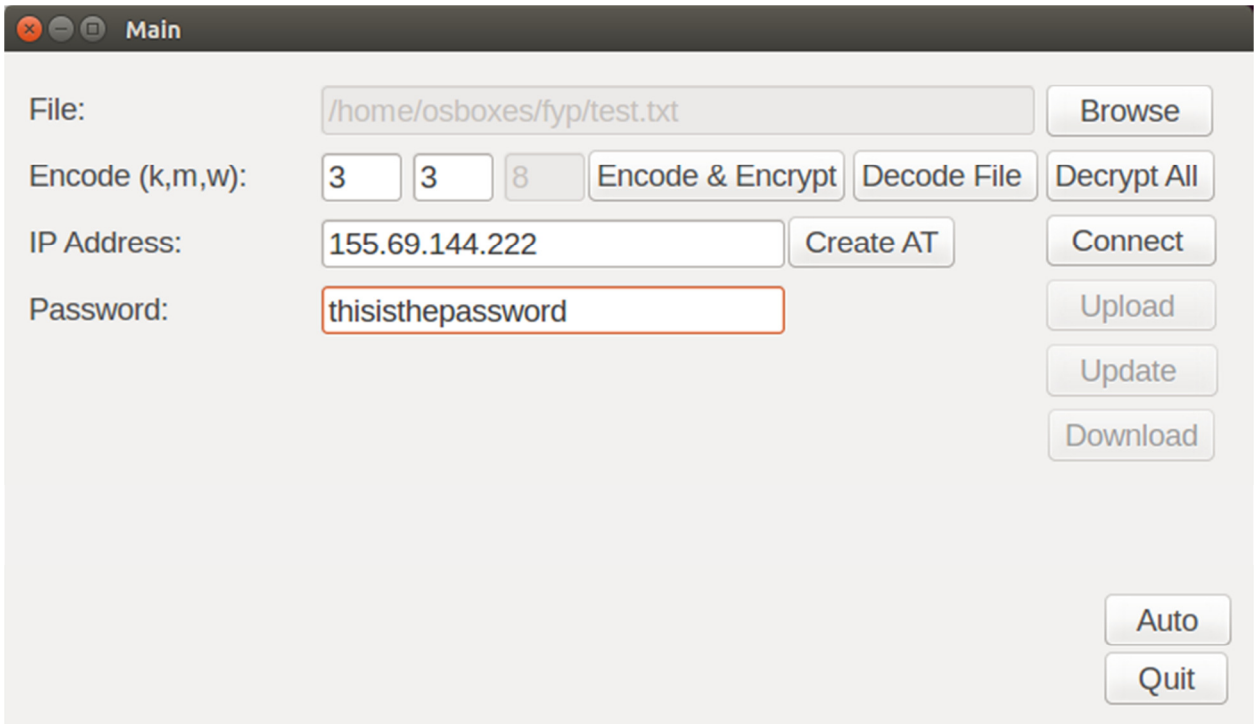
This proof-of-concept has shown that layered security on a decentralized storage is possible and feasible for personal to small business use. However, only the original owner may access and modify the file.

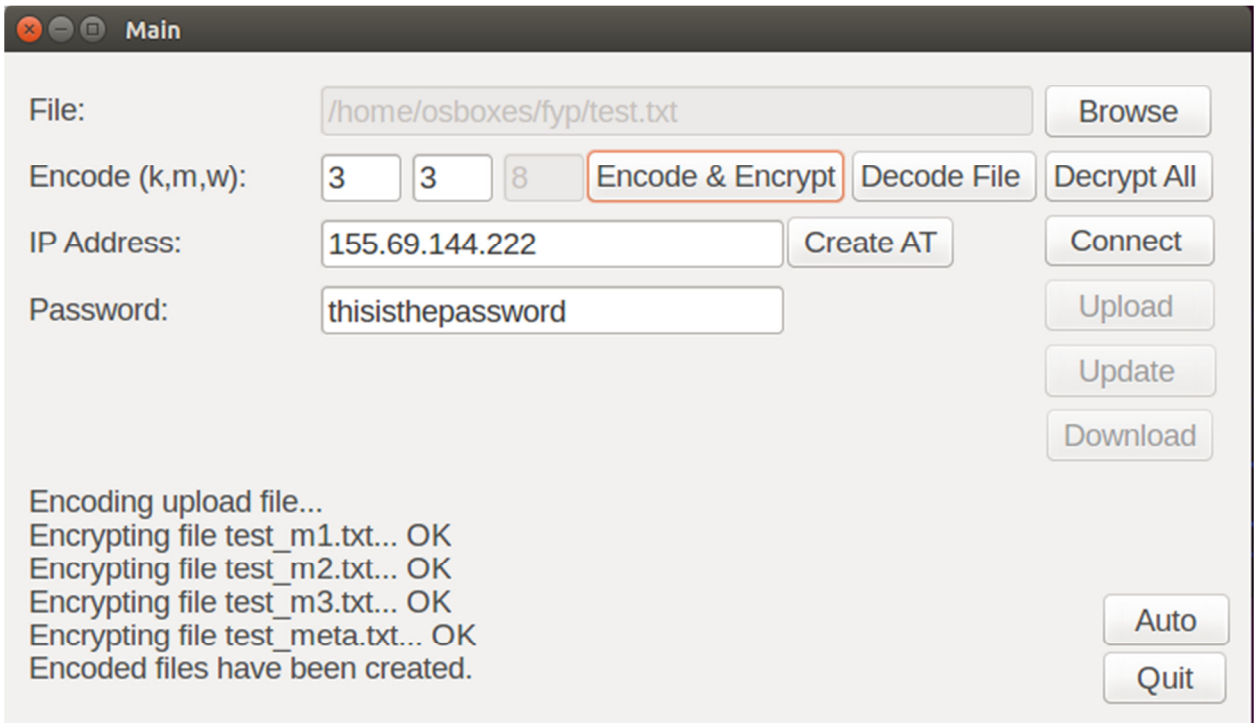
Collaboration among different people for a file that is being stored on the cloud can be added by using the Bidirectional Proxy Re-encryption scheme. This scheme consists of a few algorithms. A pair of public and private key will first be generated. From this key pair, two pairs of public and private keys are then generated relative to the first key pair. This is also known as the re-encryption key generation, which works both ways for both clients. When a message is encrypted with the private key of the first key pair, the server performs re-encryption and after which the ciphertext can be decrypted by the public key of the other pair.

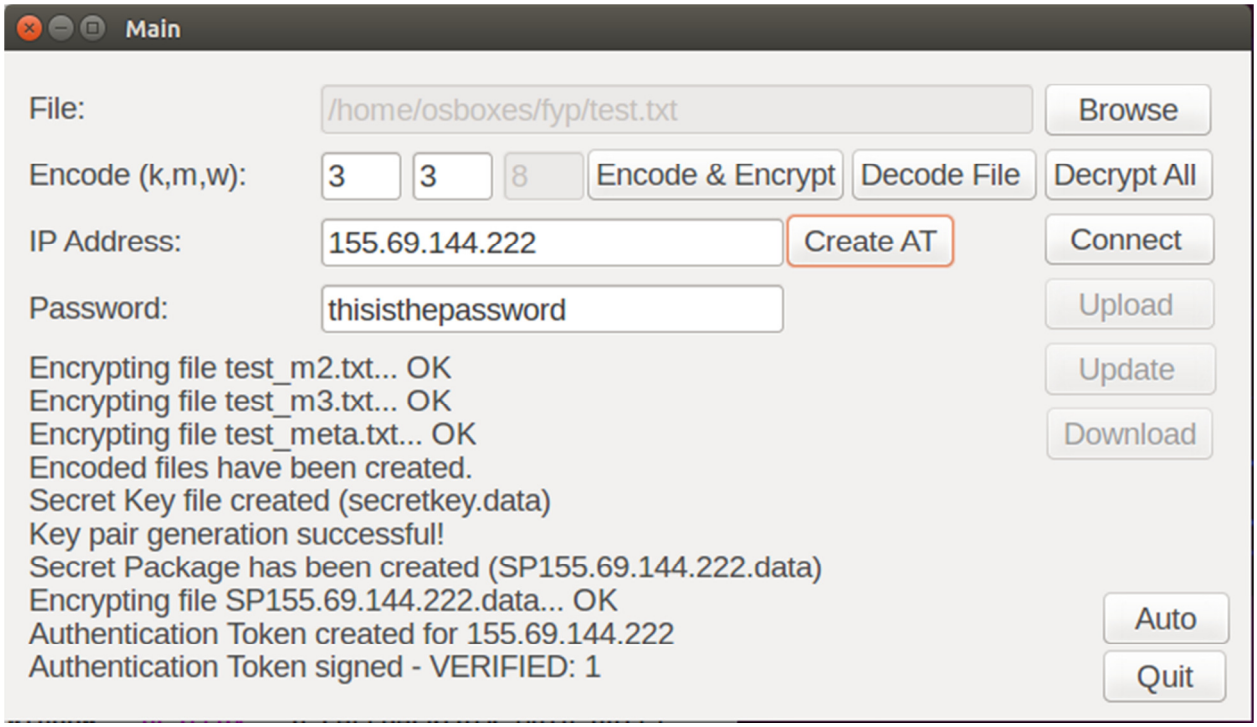
Other future works can include using a mobile application that provides a separate authenticator for the possession factor.

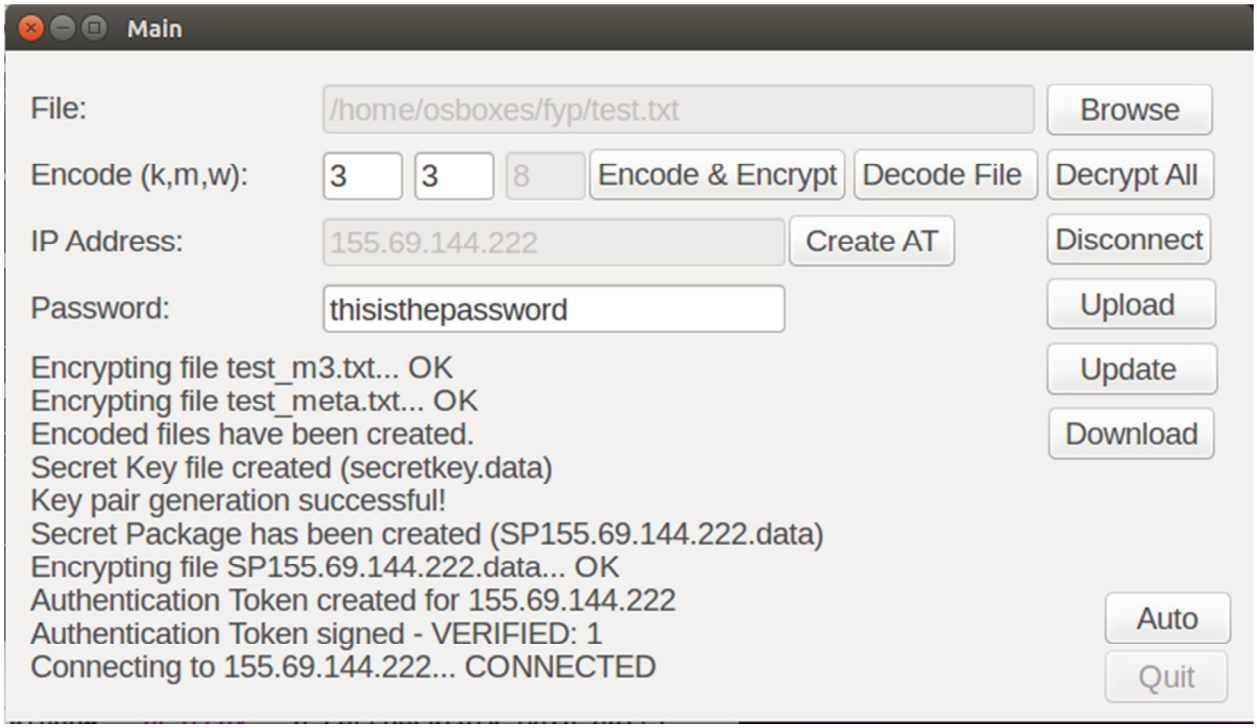
APPENDIX

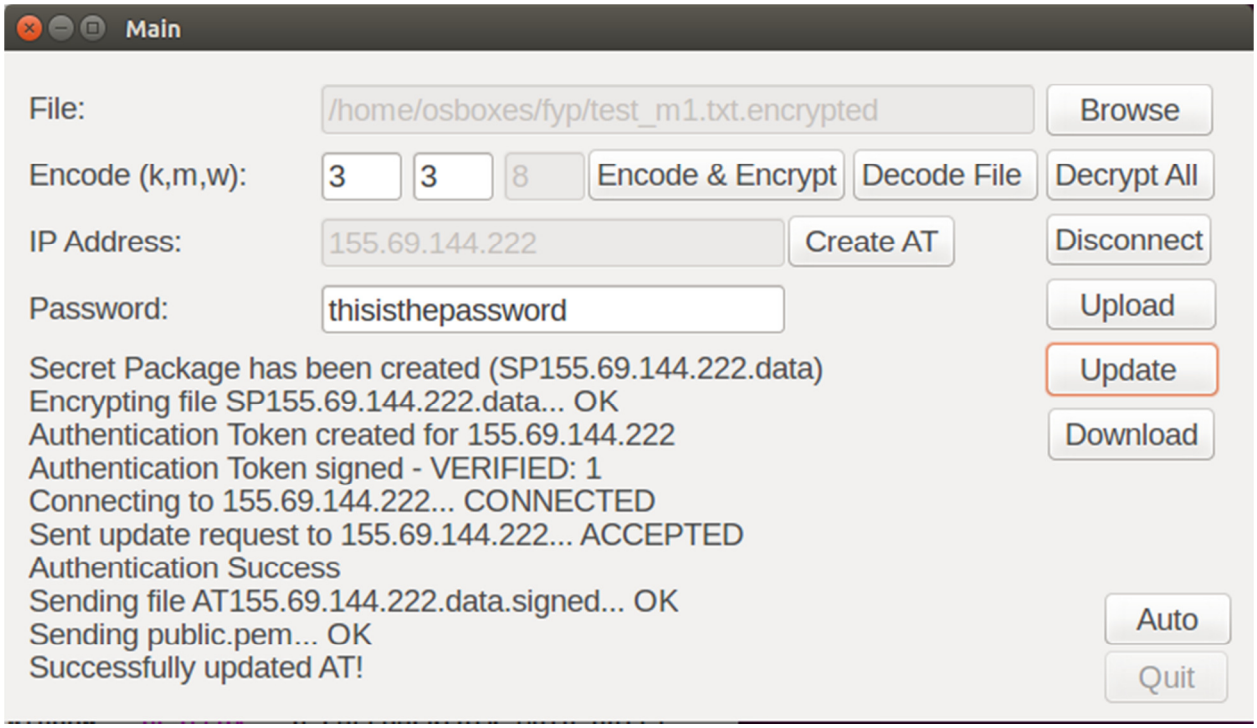
APPENDIX A

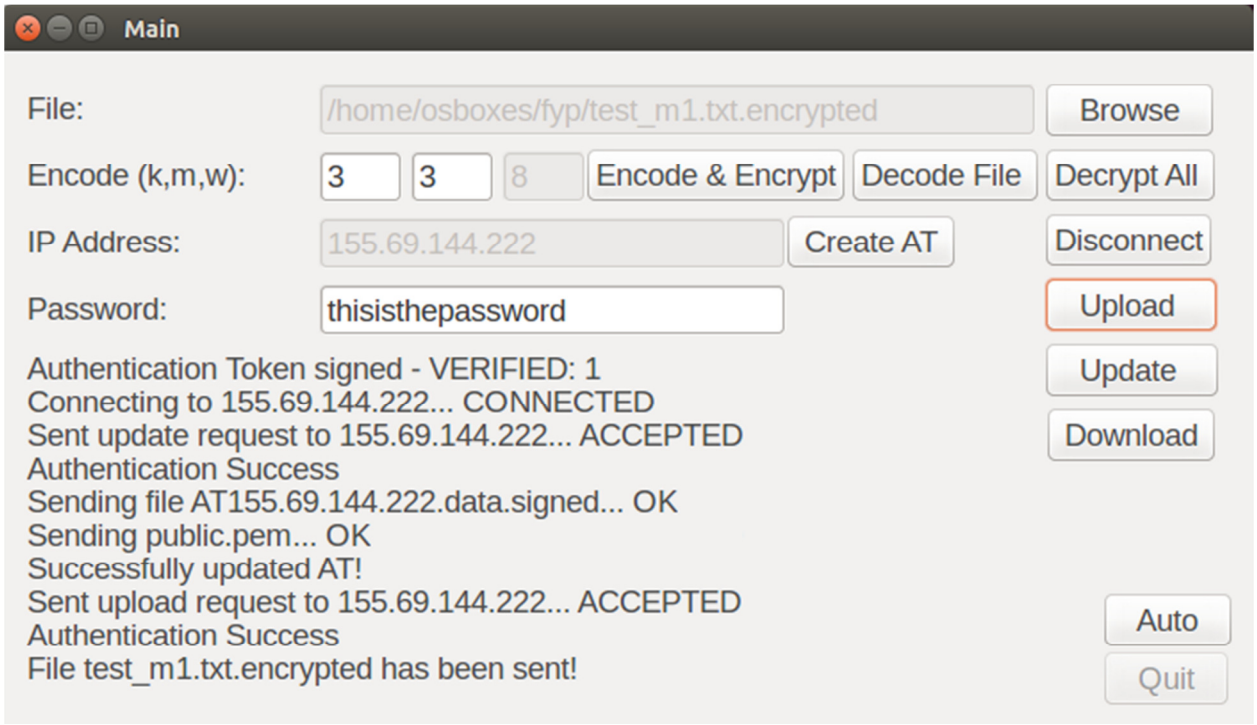


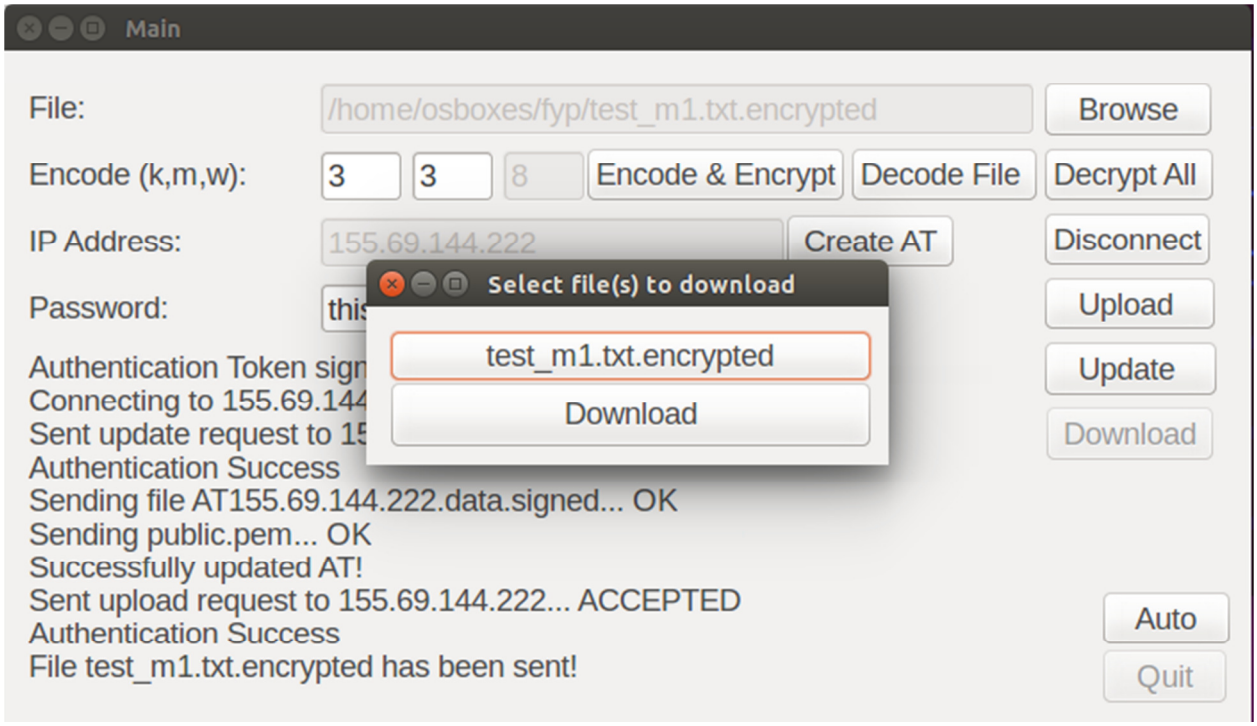


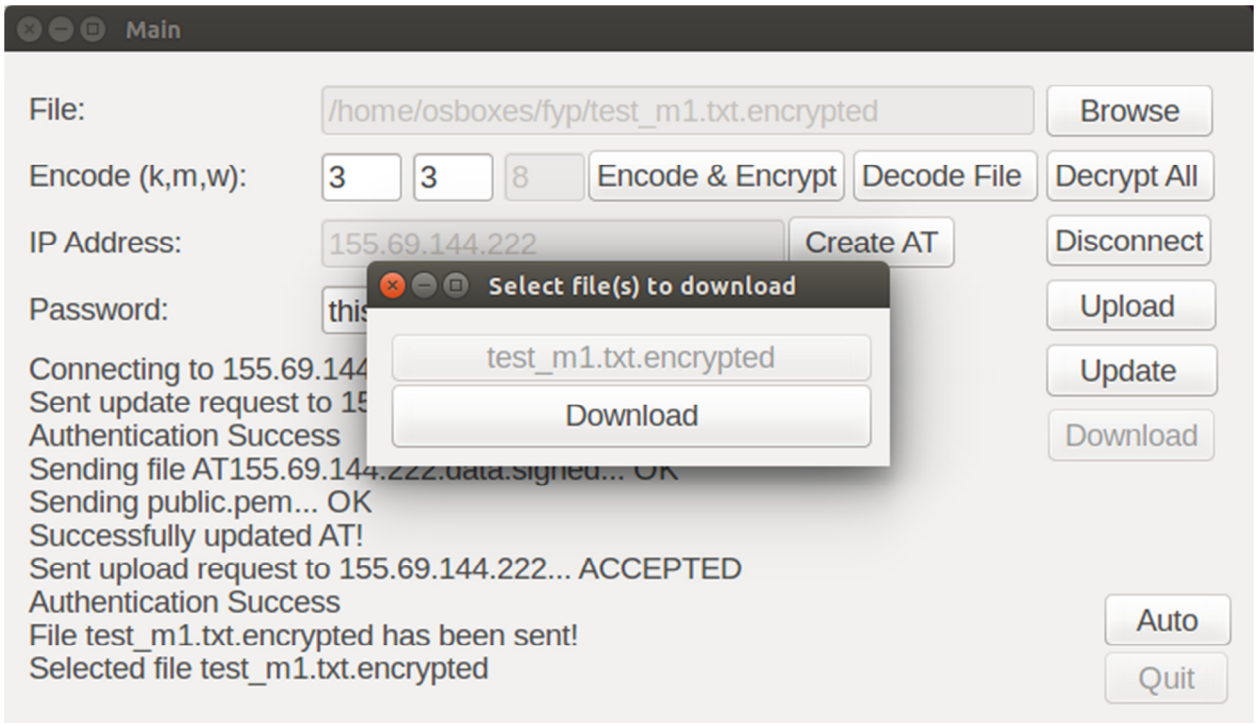

Main
File:
/home/osboxes/fyp/test.txt
Browse
Encode (k,m,w):
3
3
8
Encode & Encrypt
Decode File
Decrypt All
IP Address:
155.69.144.222
Create AT
Connect
Password:
thisisthepassword
Upload
Update
Download
Encrypting file test_m2.txt... OK
Encrypting file test_m3.txt... OK
Encrypting file test_meta.txt... OK
Encoded files have been created.
Secret Key file created (secretkey.data)
Key pair generation successful!
Secret Package has been created (SP155.69.144.222.data)
Encrypting file SP155.69.144.222.data... OK
Authentication Token created for 155.69.144.222
Authentication Token signed - VERIFIED: 1
Auto
Quit

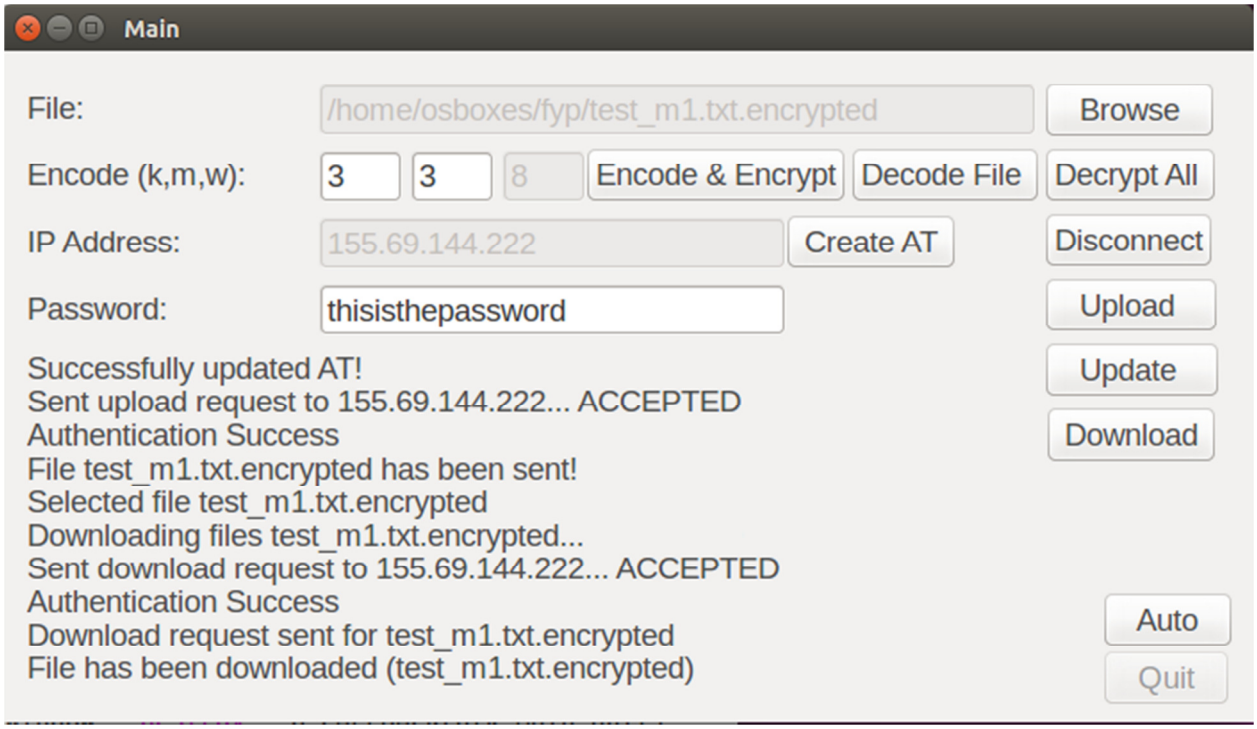

Main
File:
/home/osboxes/fyp/test.txt
Browse
Encode (k,m,w):
3
3
8
Encode & Encrypt
Decode File
Decrypt All
IP Address:
155.69.144.222
Create AT
Disconnect
Password:
thisisthepassword
Upload
Update
Download
Encrypting file test_m3.txt... OK
Encrypting file test_meta.txt... OK
Encoded files have been created.
Secret Key file created (secretkey.data)
Key pair generation successful!
Secret Package has been created (SP155.69.144.222.data)
Encrypting file SP155.69.144.222.data... OK
Authentication Token created for 155.69.144.222
Authentication Token signed - VERIFIED: 1
Connecting to 155.69.144.222... CONNECTED
Auto
Quit

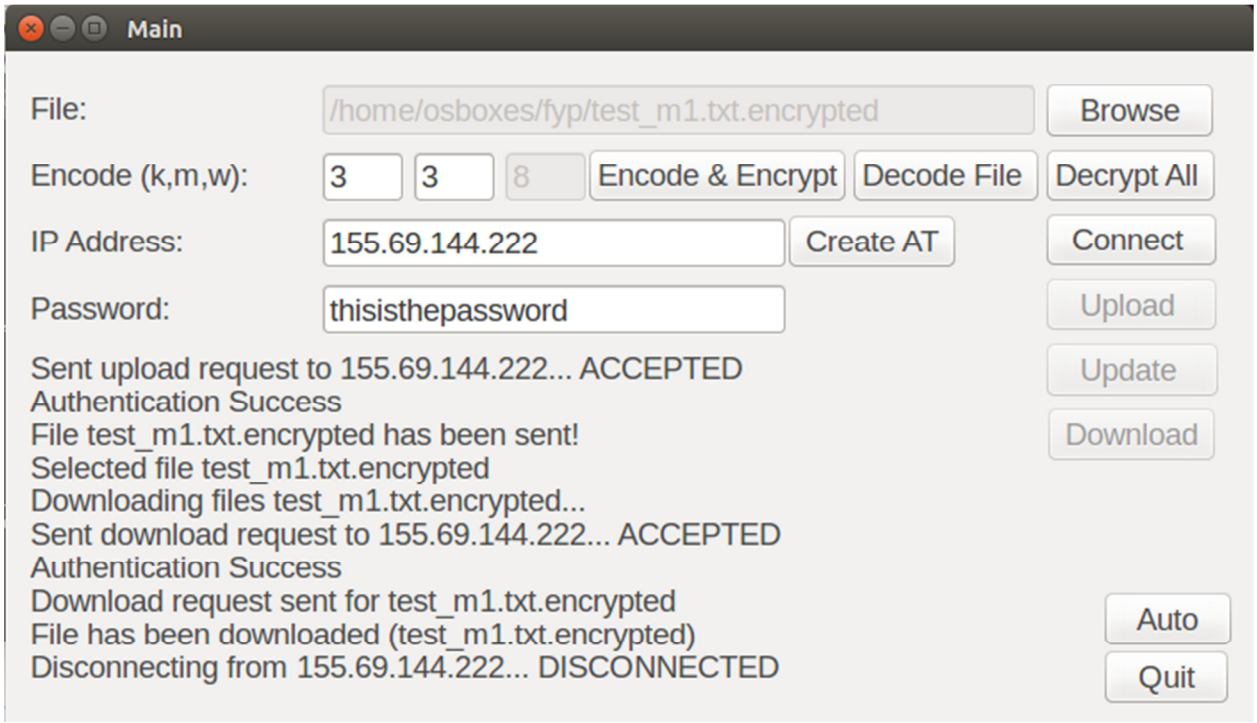

Main
File:
/home/osboxes/fyp/test_m1.txt.encrypted
Browse
Encode (k,m,w):
3
3
8
Encode & Encrypt
Decode File
Decrypt All
IP Address:
155.69.144.222
Create AT
Disconnect
Password:
thisisthepassword
Upload
Secret Package has been created (SP155.69.144.222.data)
Encrypting file SP155.69.144.222.data... OK
Authentication Token created for 155.69.144.222
Authentication Token signed - VERIFIED: 1
Connecting to 155.69.144.222... CONNECTED
Sent update request to 155.69.144.222... ACCEPTED
Authentication Success
Sending file AT155.69.144.222.data.signed... OK
Sending public.pem... OK
Successfully updated AT!
Update
Download
Auto
Quit

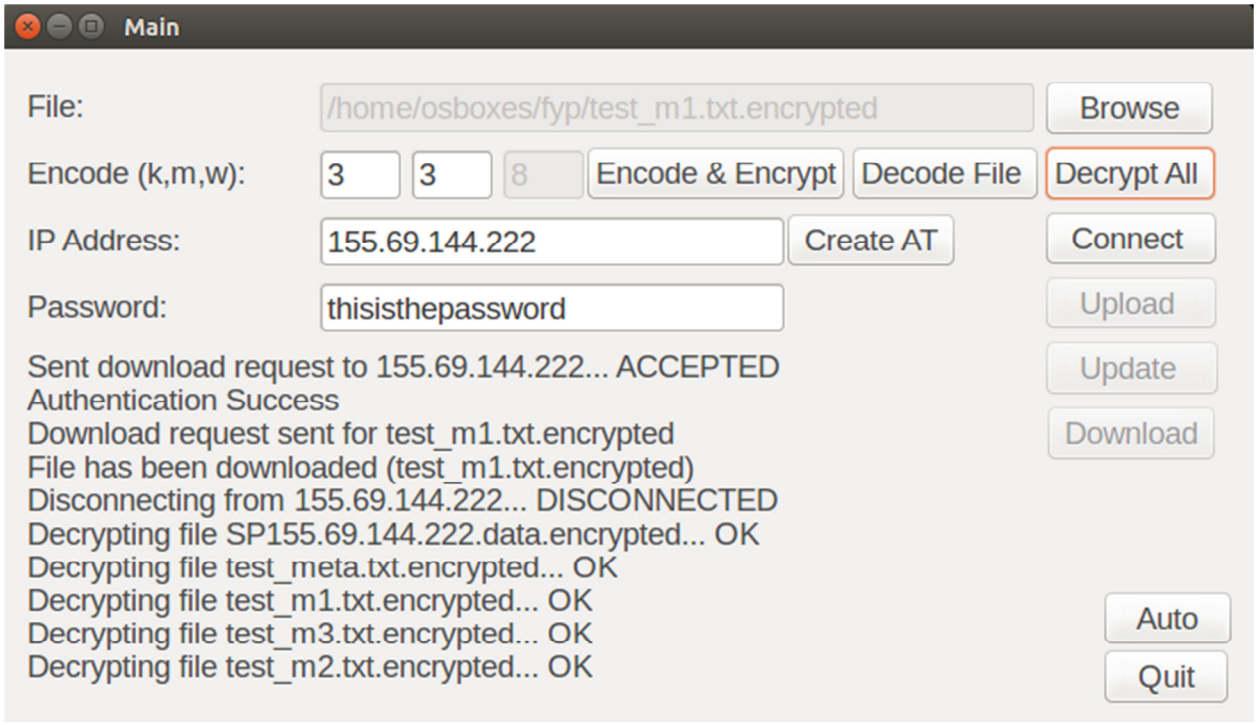

Main
File:
/home/osboxes/fyp/test_m1.txt.encrypted
Browse
Encode (k,m,w):
3
3
8
Encode & Encrypt
Decode File
Decrypt All
IP Address:
155.69.144.222
Create AT
Disconnect
Password:
thisisthepassword
Upload
Authentication Token signed - VERIFIED: 1
Connecting to 155.69.144.222... CONNECTED
Sent update request to 155.69.144.222... ACCEPTED
Authentication Success
Sending file AT155.69.144.222.data.signed... OK
Sending public.pem... OK
Successfully updated AT!
Sent upload request to 155.69.144.222... ACCEPTED
Authentication Success
File test_m1.txt.encrypted has been sent!
Update
Download
Auto
Quit

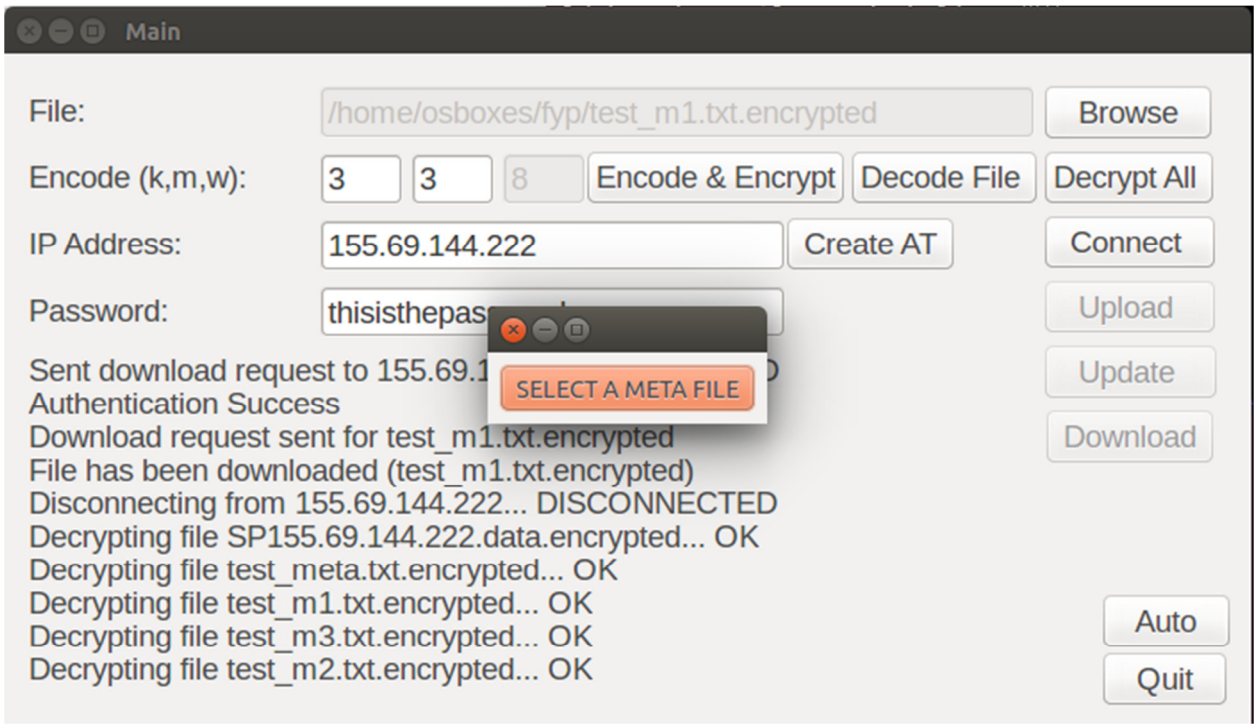

Main
File:
/home/osboxes/fyp/test_m1.txt.encrypted
Browse
Encode (k,m,w):
3
3
8
Encode & Encrypt
Decode File
Decrypt All
IP Address:
155.69.144.222
Create AT
Disconnect
Password:
Upload
Update
Download
Select file(s) to download
test_m1.txt.encrypted
Download
Authentication Token sign
Connecting to 155.69.144
Sent update request to 15
Authentication Success
Sending file AT155.69.144.222.data.signed... OK
Sending public.pem... OK
Successfully updated AT!
Sent upload request to 155.69.144.222... ACCEPTED
Authentication Success
File test_m1.txt.encrypted has been sent!
Auto
Quit

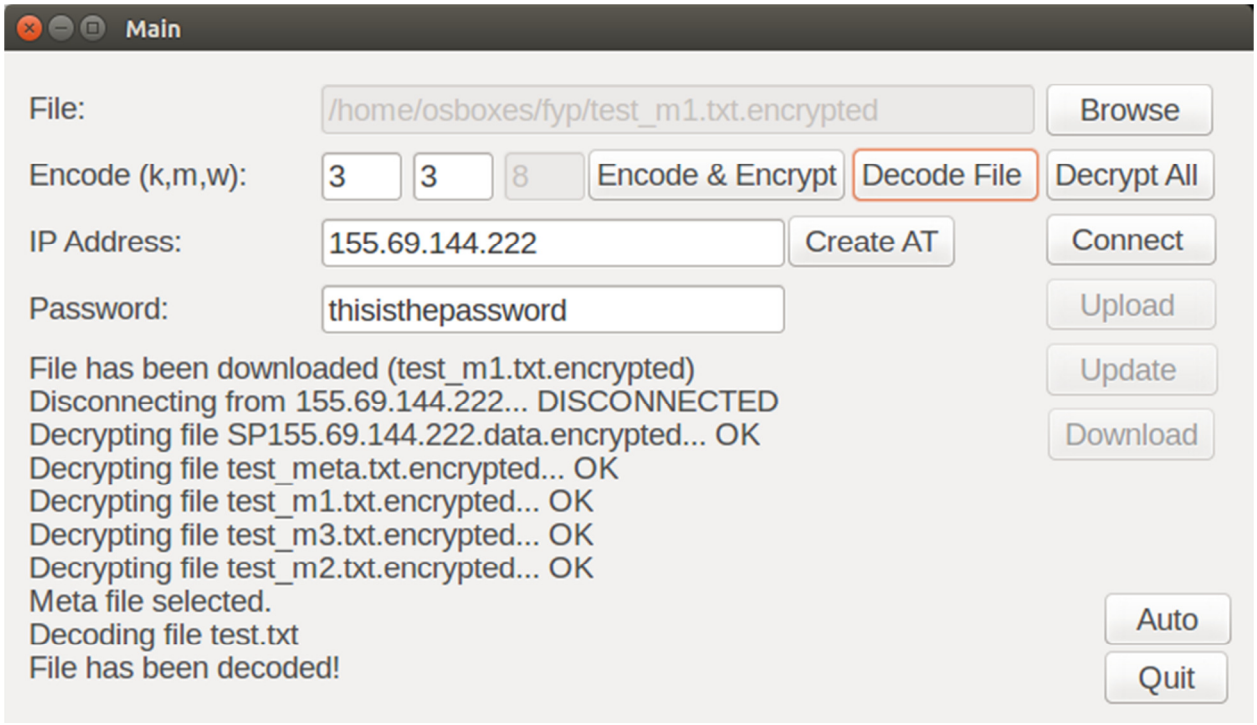

Main
File:
/home/osboxes/fyp/test_m1.txt.encrypted
Browse
Encode (k,m,w):
3
3
8
Encode & Encrypt
Decode File
Decrypt All
IP Address:
155.69.144.222
Create AT
Disconnect
Password:
Upload
Update
Download
Select file(s) to download
test_m1.txt.encrypted
Download
Connecting to 155.69.144
Sent update request to 15
Authentication Success
Sending public.pem... OK
Successfully updated AT!
Sent upload request to 155.69.144.222... ACCEPTED
Authentication Success
File test_m1.txt.encrypted has been sent!
Selected file test_m1.txt.encrypted
Auto
Quit


Main
File:
/home/osboxes/fyp/test_m1.txt.encrypted
Browse
Encode (k,m,w):
3
3
8
Encode & Encrypt
Decode File
Decrypt All
IP Address:
155.69.144.222
Create AT
Disconnect
Password:
thisisthepassword
Upload
Update
Download
Auto
Quit
Successfully updated AT!
Sent upload request to 155.69.144.222... ACCEPTED
Authentication Success
File test_m1.txt.encrypted has been sent!
Selected file test_m1.txt.encrypted
Downloading files test_m1.txt.encrypted...
Sent download request to 155.69.144.222... ACCEPTED
Authentication Success
Download request sent for test_m1.txt.encrypted
File has been downloaded (test_m1.txt.encrypted)


Main
File:
/home/osboxes/fyp/test_m1.txt.encrypted
Browse
Encode (k,m,w):
3
3
8
Encode & Encrypt
Decode File
Decrypt All
IP Address:
155.69.144.222
Create AT
Connect
Password:
thisisthepassword
Upload
Update
Download
Auto
Quit
Sent upload request to 155.69.144.222... ACCEPTED
Authentication Success
File test_m1.txt.encrypted has been sent!
Selected file test_m1.txt.encrypted
Downloading files test_m1.txt.encrypted...
Sent download request to 155.69.144.222... ACCEPTED
Authentication Success
Download request sent for test_m1.txt.encrypted
File has been downloaded (test_m1.txt.encrypted)
Disconnecting from 155.69.144.222... DISCONNECTED


Main
File:
/home/osboxes/fyp/test_m1.txt.encrypted
Browse
Encode (k,m,w):
3
3
8
Encode & Encrypt
Decode File
Decrypt All
IP Address:
155.69.144.222
Create AT
Connect
Password:
thisisthepassword
Upload
Update
Download
Sent download request to 155.69.144.222... ACCEPTED
Authentication Success
Download request sent for test_m1.txt.encrypted
File has been downloaded (test_m1.txt.encrypted)
Disconnecting from 155.69.144.222... DISCONNECTED
Decrypting file SP155.69.144.222.data.encrypted... OK
Decrypting file test_meta.txt.encrypted... OK
Decrypting file test_m1.txt.encrypted... OK
Decrypting file test_m3.txt.encrypted... OK
Decrypting file test_m2.txt.encrypted... OK
Auto
Quit


Main
File:
/home/osboxes/fyp/test_m1.txt.encrypted
Browse
Encode (k,m,w):
3
3
8
Encode & Encrypt
Decode File
Decrypt All
IP Address:
155.69.144.222
Create AT
Connect
Password:
Upload
SELECT A META FILE
Update
Download
Authentication Success
Download request sent for test_m1.txt.encrypted
File has been downloaded (test_m1.txt.encrypted)
Disconnecting from 155.69.144.222... DISCONNECTED
Decrypting file SP155.69.144.222.data.encrypted... OK
Decrypting file test_meta.txt.encrypted... OK
Decrypting file test_m1.txt.encrypted... OK
Decrypting file test_m3.txt.encrypted... OK
Decrypting file test_m2.txt.encrypted... OK
Auto
Quit


Main
File:
/home/osboxes/fyp/test_m1.txt.encrypted
Browse
Encode (k,m,w):
3
3
8
Encode & Encrypt
Decode File
Decrypt All
IP Address:
155.69.144.222
Create AT
Connect
Password:
thisisthepassword
Upload
Update
Download
File has been downloaded (test_m1.txt.encrypted)
Disconnecting from 155.69.144.222... DISCONNECTED
Decrypting file SP155.69.144.222.data.encrypted... OK
Decrypting file test_meta.txt.encrypted... OK
Decrypting file test_m1.txt.encrypted... OK
Decrypting file test_m3.txt.encrypted... OK
Decrypting file test_m2.txt.encrypted... OK
Meta file selected.
Decoding file test.txt
File has been decoded!
Auto
Quit