

## Project Report

Secure Text Transfer Using Diffie-Hellman Key Exchange Based On Cloud

Parth Trehan (2010660)  
Hardik Gaur (2010638)

Supervisor: Dr. Sachin Sharma  
Dr. Preeti Mishra

Department of Computer Science and Engineering  
Graphic Era deemed to be a University

July 19, 2018

---

## **Abstract**

Advent of Cloud Computing has been a phenomenal phase in the history of computer science. It provided capabilities to solve many problems that were earlier deemed impossible to be computed by a machine. It removed the pressure from those responsible for manufacturing better machines to keep up with the increasing complexity of the problems that the machines are intended to solve. Cloud Computing provided platform for better utilization of the resource spread across the world. Being a nascent field, it is crowded with many different problems that the engineers and scientist are working assiduously to eliminate. One of the main drawbacks with cloud is security. So, this project proposes a mechanism for secure file storage cloud using encryption and Diffie Hellman. The algorithm involves encrypting the file stored on the cloud and using Diffie Hellman for authenticating the user to decrypt the required file.

The implementation and building process of the code can be found [here](#)

## **Table of Contents**

1.0 Introduction.....	1
2.0 Diffie-Hellman Key Exchange.....	1
2.1 Prime Number.....	2
2.2 Method.....	2
3.0 Encryption.....	3
3.1 Advanced Encryption Standard (AES).....	3
4.0 Python (programming language) .....	4
4.1 Python-tkinter.....	4
4.2 Python-flask.....	4
4.3 Python-crypto.....	5
5.0 Amazon Web Services.....	6
5.1AWS EC-2.....	6
6.0 Implementation.....	6
6.1 Planning and analysis.....	7
6.2 Stand-alone-application.....	7
6.2.1 Diffie-Hellman.....	7
6.2.2 Encrytion.....	7
6.2.3 GUI.....	7
6.3 Web-application.....	7
6.4 Hosting on Amazon AWS-EC2.....	8
7.0 Conclusion.....	12
8.0 References.....	13

## 1.0 Introduction

Cloud security is one of the main concerns in the cloud computing domain. Storing personal and sensitive information on a third-party storage medium poses serious risks of data theft and data misuse by any person with malicious intent. The threat is so humongous that it has dissuaded governments and many other big organizations from migrating their operations on a cloud platform. The traditional methods of securing files and information are superfluous in the scenario of cloud. Extensive research and study is undergoing in this field to make cloud more secure and reliable. Among this behemoth instances of research, some of the methods that stand out include AES encryption and Diffie Hellman Key Exchange. The latter method is so powerful that it may take millions of years for even the most powerful computers of current times to crack the code and reads the file. Our approach proposes a method that involves encrypting the file using any standard encryption technique and using Diffie Hellman for user authentication. In this way the files can be saved in a public domain securely without the threat of being used by any unauthorised person.

## 2.0 Diffie Hellman Key Exchange

Diffie–Hellman key exchange (DH) is a method of securely exchanging cryptographic keys over a public channel and was one of the first public-key protocols named after Whitfield Diffie and Martin Hellman.<sup>[1]</sup> DH is one of the earliest practical examples of public key exchange implemented within the field of cryptography.

In public key cryptosystem, enciphering and deciphering are governed by distinct keys, E and D, such that computing D from E is computationally infeasible (e.g., requiring more than  $10^{100}$  instructions). The enciphering key E can thus be publicly disclosed without compromising the deciphering key D. This was the main ideology behind Diffie-Hellman Key Exchange Protocol. Each user of the network can, therefore, place his enciphering key in a public directory. This enables any user of the system to send a message to any other user enciphered in such a way that only the intended receiver can decipher it. As such, a public key cryptosystem is a multiple access cipher. A private conversation can therefore be held between any two individuals regardless of whether they have ever communicated before. Each one sends messages to the other enciphered in the receiver's public enciphering key and deciphers the messages he receives using his own secret deciphering key.

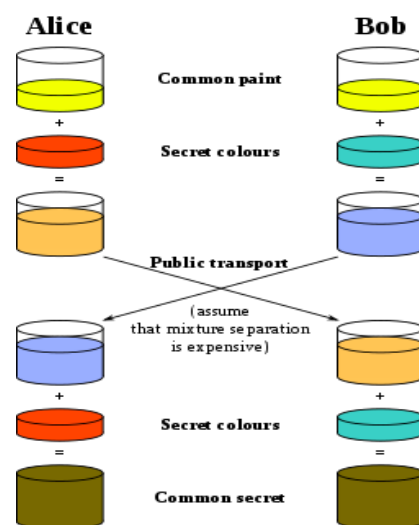


Figure 1: Illustration of idea behind Diffie-Hellman

Diffie–Hellman key exchange establishes a shared secret between two parties that can be used for secret communication for exchanging data over a public network. The above conceptual diagram illustrates the general idea of the key exchange by using colours instead of very large numbers. The process begins by having the two parties, Alice and Bob, agree on an arbitrary starting colour that does not need to be kept secret in this example the color is yellow. Each of them selects a secret color that they keep to themselves – in this case, orange and blue-green. The crucial part of the process is that Alice and Bob each mix their own secret color together with their mutually shared color, resulting in orange-tan and light-blue mixtures respectively, and then publicly exchange the two mixed colours. Finally, each of the two mixes the color he or she received from the partner with his or her own private color. The result is a final color mixture (yellow-brown in this case) that is identical to the partner's final color mixture. If a third party listened to the exchange, it would be computationally difficult for this party to determine the secret colors. In fact, when using large numbers rather than colors, this action is computationally expensive for modern supercomputers to do in a reasonable amount of time. <sup>[2]</sup>

## 2.1 Prime Number

A prime number (or a prime) is a natural number greater than 1 that cannot be formed by multiplying two smaller natural numbers.

The only user-defined pre-existing parameter in the Diffie-Hellman protocol is the selection of prime number. The prime number  $p$  should be large enough to defend against the known attacks against it. The most efficient attack is NFS (attack on the network file system); that has been used against numbers on the order of  $2^{768}$  (a 232-digit number). It would appear wise to pick a  $p$  that's considerably bigger than that; around 1024 bits at a minimum, and more realistically at least 1536 bits. Another property about  $p$  is that  $p-1$  should have a large prime factor  $q$ , and one should know what the factorization of  $p-1$  is. If we pick a random prime  $p$ , and a random generator  $g$ , well, we're probably secure, but we won't be certain (and we might leak a few bits of the private exponent if the order of your random  $g$  happens to have some small factors).

## 2.2 Method

Follow the mathematical implementation of Diffie Hellman key exchange protocol.

$p$  is a prime number.

$g$  is a primitive root modulo of  $p$

1. Alice and Bob agree to use a modulus  $p = 23$  and base  $g = 5$
2. Alice gets her private key (key which she should not share with anyone) generated as 4.
3. Thus, public key generated for Alice shall be  $5^4 \% 23 = 625 \% 23 = 4$
4. Bob gets his private key (key which he should not share with anyone) generated as 3.
5. Thus, public key generated for Bob shall be  $5^3 \% 23 = 125 \% 23 = 10$
6. Now, Alice gets the public key of Bob and generates a secret key. i.e.  
(public key of Bob <sup>Private Key of Alice</sup>) mod  $p$   
 $\Rightarrow (10^4) \% 23 \Rightarrow 10000 \% 23 \Rightarrow 18$
7. On the other side, Bob also uses a similar method to generate a secret key i.e.  
(public key of Alice <sup>Private Key of Bob</sup>) mod  $p$   
 $\Rightarrow (4^3) \% 23 \Rightarrow 64 \% 23 \Rightarrow 18$

Thus, it is proven that mathematically, Alice and Bob generate the same key without each one of them knowing other one's private key. This is the implementation of Diffie-Hellman Key Exchange Protocol.

### 3.0 Encryption

Encryption is widely used on the internet to protect user information being sent between a browser and a server, including passwords, payment information and other personal information that should be considered private. Organizations and individuals also commonly use encryption to protect sensitive data stored on computers, servers and mobile devices like phones or tablets. There are various encryption techniques that are present some of which are:

- Triple DES
- Blowfish
- RSA
- Twofish
- AES

The technique that we have used in our project is AES and it is described below.

### 3.1 Advanced Encryption Standard

The more popular and widely adopted symmetric encryption algorithm nowadays is the Advanced Encryption Standard (AES). It is found to be at least six time faster than triple DES.

A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback, but it was found to be slow.

The AES has three fixed 128-bit block ciphers with cryptographic key sizes of 128, 192 and 256 bits. Key size is unlimited, whereas the block size maximum is 256 bits. The AES design is based on a substitution-permutation network (SPN) and does not use the Data Encryption Standard (DES) Feistel network. The diagram below shows the implementation of AES encryption technique.

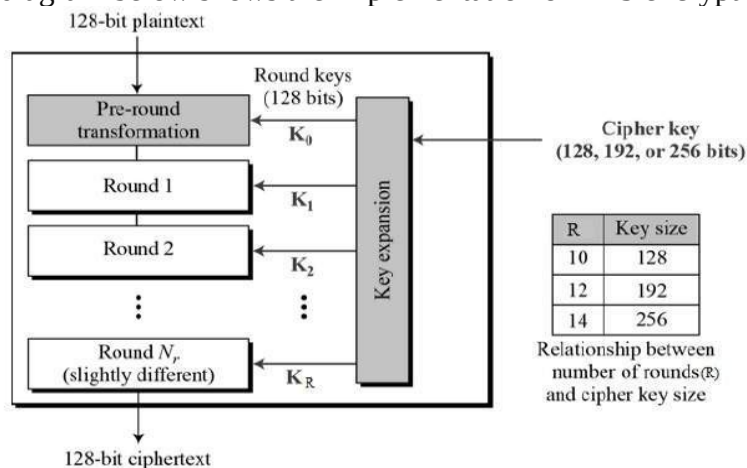


Figure 2: The schematic of AES structure

## 4.0 Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespaces. It provides constructs that enable clear programming on both small and large scales.<sup>[3]</sup> Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative and procedural, and has a large and comprehensive standard library.

Such large and comprehensive standard libraries along with the documented support helped us to choose python as the language which we used to build both, the stand alone as well as web-based application.

Below is a brief description of the frameworks and libraries extensively used to build the desired framework.

### 4.1 Python tkinter

Tkinter is Python's de-facto standard GUI (Graphical User Interface) package. It is a thin object-oriented layer on top of Tcl/Tk.<sup>[5]</sup> Tkinter is not the only GUI Programming toolkit for Python. It is however the most commonly used one. Cameron Laird calls the yearly decision to keep Tkinter "one of the minor traditions of the Python world." Tkinter is a GUI (graphical user interface) widget set for Python. This document was written for Python 2.7 and Tkinter 8.5 running in the X Window system under Linux. Tkinter helps users to build a cross-platform application and is easy to use, thus, we used it to build the GUI of our stand-alone application. Figure 3 shows the GUI of **thrain**, the framework of the hour.

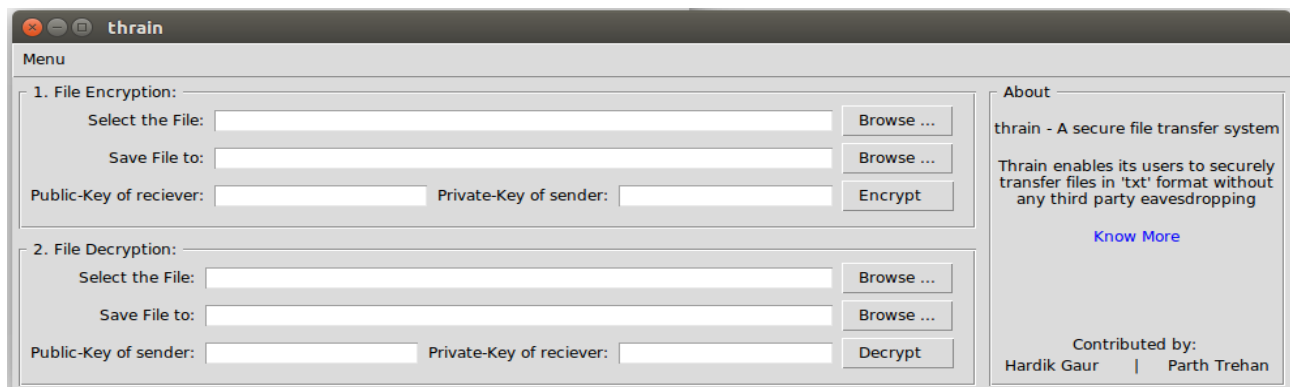


Figure 3: GUI built using python-tkinter

### 4.2 Python flask

Flask is a BSD licensed microframework for Python based on Werkzeug and Jinja 2. "Micro" does not mean that your whole web application must fit into a single Python file (although it certainly can), nor does it mean that Flask is lacking in functionality. The "micro" in microframework means Flask aims to keep the core simple but extensible. Flask won't make many decisions for you, such as what database to use. Those decisions that it does make, such as what templating engine to use, are easy to change. Everything else is up to you, so that Flask can be everything you need and nothing you don't.

By default, Flask does not include a database abstraction layer, form validation or anything else where different libraries already exist that can handle that. Instead, Flask supports extensions to add such functionality to your application as if it was implemented in Flask itself. Numerous extensions provide database integration, form validation, upload handling, various open authentication technologies, and more. Flask may be “micro”, but it’s ready for production use on a variety of needs. Flask has many configuration values, with sensible defaults, and a few conventions when getting started. By convention, templates and static files are stored in subdirectories within the application’s Python source tree, with the names templates and static respectively. While this can be changed, you usually don’t have to, especially when getting started.

### 4.3 Python crypto

Another python’s extensively maintained library is PyCrypto. It is an actively developed library that provides cryptographic recipes and primitives. It provides secure hash functions and various encryption algorithms. Hash functions played an important role in the formation of the framework. The main function of the hash function was to generate a private key for new users, which was less than the pre-defined prime number. Library hashlib was deployed for the cause, where a variable length was being passed to the function and a digest was generated. This library was also useful as it already had implementation of various symmetric- and asymmetric-key encryption algorithms like AES, blowfish, ARC2 and many more. Encryption algorithms transform plaintext in some way that is dependent on a key or key pair, producing ciphertext. We employed AES and blowfish (symmetric-key encryption) for encrypting the text. We performed a double layered encryption. Encryption here could be used in tailor made format. Below are the snippets 1 and 2 depicting the python code to encrypt text using AES and blowfish respectively.

```
>>> from Crypto.Cipher import AES
>>> from Crypto import Random
>>> key = b'Sixteen byte key'
>>> iv = Random.new().read(AES.block_size)
>>> cipher = AES.new(key, AES.MODE_CFB, iv)
>>> msg = iv + cipher.encrypt(b'Attack at dawn')
```

Snippet 1: Python snippet to encrypt a text using AES

```
>>> from Crypto.Cipher import Blowfish
>>> from Crypto import Random
>>> from struct import pack
>>> bs = Blowfish.block_size
>>> key = b'An arbitrarily long key'
>>> iv = Random.new().read(bs)
>>> cipher = Blowfish.new(key, Blowfish.MODE_CBC, iv)
>>> plaintext = b'docendo discimus '
>>> plen = bs - divmod(len(plaintext),bs)[1]
>>> padding = [plen]*plen
>>> padding = pack('b'*plen, *padding)
>>> msg = iv + cipher.encrypt(plaintext + padding)
```

Snippet 2: Python snippet to encrypt a text using Blowfish



## **5.0 Amazon Web Services**

Amazon Web Services (AWS) is a subsidiary of Amazon.com that provides on-demand cloud computing platforms to individuals, companies and governments, on a paid subscription basis. The technology allows subscribers to have at their disposal a virtual cluster of computers, available all the time, through the Internet. AWS's version of virtual computers emulate most of the attributes of a real computer including hardware (CPU(s) & GPU(s) for processing, local/RAM memory, hard-disk/SSD storage); a choice of operating systems; networking; and pre-loaded application software such as web servers, databases, CRM, etc. Each AWS system also virtualizes its console I/O (keyboard, display, and mouse), allowing AWS subscribers to connect to their AWS system using a modern browser.<sup>[2]</sup>

### **5.1 Elastic Compute Cloud [EC2]**

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.

Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change. Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use. Amazon EC2 provides developers the tools to build failure resilient applications and isolate them from common failure scenarios. For compute, AWS' main offering is its EC2 instances, which can be tailored with a large number of options. It also provides related services such as Elastic Beanstalk for app deployment, the EC2 Container service, AWS Lambda and Autoscaling.

In general terms, prices are roughly comparable, especially since AWS shifted from by-the-hour to by-the-second pricing for its EC2 and EBS services in 2017. This economic pricing, reliable and secure infrastructure and vast online support enabled us to use Amazon to host and deploy the service on its IaaS platform.

## **6.0 Implementation**

This section describes the methodologies and implementation details of the targeted framework. The contents in this section are organized as follows: *Section 7.1* contains the details of the planning done before executing the targeted framework. We have even discussed the possible risks and issues that could have occurred in other methodologies and the reason why we chose this implementation. We concluded that the safest mode of transfer and encryption is when we encrypted the text offline and used the cloud services as an online directory to store keys and encrypted documents. *Section 7.2* describes the details of the stand-alone application and the details how we build the application from scratch. *Section 7.3* enlists the method we used to build the online directory and hosted it. The code files and related documents can be found on our GitHub account.<sup>[7]</sup>

## 6.1 Planning and Analysis

The main task of this project was to provide as secure a file storage on the cloud as possible. So, several issues had to be sorted out like :

- Where the file needs to be encrypted.?
- How the user must be authenticated?
- What will be the key for AES encryption?
- How this key will be related to the Diffie Hellman Key exchange protocol?

We first thought of encrypting the text online, but the *attack man in the middle* made us not choose that method. We also learnt that about an attack abbreviated as NFS. NFS in an attack in which a key of order  $2^{768}$  could be computed. This was approximately 232 digits. Thus, we came to a conclusion that a larger prime number is needed in the process. Therefore, we used a prime number having 600 digits.

While analysing all these questions we came upon this course of action. To provide greater control to the owner of the file we encrypted the file on the owner's computer itself using an application. The Diffie Hellman was used to generate the file and only those users who have the final same key from this process would be able to decrypt the file. The final key generated from Diffie Hellman, being same for both intended participants, was used as the basis for the key for AES encryption. The following sub-sections describe the different parts of the project in greater detail.

## 6.2 Stand-alone-application (thrain/src/stand-alone-application)

Stand-alone application was built to encrypt and decrypt the text using various symmetric encryption algorithms. It is a GUI based application which uses sender's key and receiver's key to encrypt and decrypt the text. This text is later uploaded on the online directory.

### 6.2.1 Diffie-Hellman (thrain/src/stand-alone-application/DH.py)

This section discussed the implementation of the Diffie-Hellman algorithm. The working and background details about the algorithm are already described in *Section 3.0* of the report. This module had four methods to execute. The tasks were

Generate a private key of given length for a new user

Generate a public key for a user using his private key

Generate secret key based on a given public and private key

First, we needed a prime number. We already discussed the NFS attack in section 7.1 and need of a strong prime number in section 3.1 of the document. Thus, we hard coded the prime number

```
0xFFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BB  
EA63B139B22514A08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C  
245E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F  
24117C4B1FE649286651ECE45B3DC2007CB8A163BF0598DA48361C55D39A69163FA8FD24  
CF5F83655D23DCA3AD961C62F356208552BB9ED529077096966D670C354E4ABC9804F1746  
C08CA18217C32905E462E36CE3BE39E772C180E86039B2783A2EC07A28FB5C55DF06F4C5  
2C9DE2BCBF6955817183995497CEA956AE515D2261898FA051015728E5A8AAAC42DAD331  
70D04507A33A85521ABDF1CBA64ECFB850458DBEF0A8AEA71575D060C7DB3970F85A6E  
1E4C7ABF5AE8CDB0933D71E8C94E04A25619DCEE3D2261AD2EE6BF12FFA06D98A0864D  
87602733EC86A64521F2B18177B200CBBE117577A615D6C770988C0BAD946E208E24FA074  
E5AB3143DB5BFCE0FD108E4B82D120A92108011A723C12A787E6D788719A10BDBA5B269  
9C327186AF4E23C1A946834B6150BDA2583E9CA2AD44CE8DBBBC2DB04DE8EF92E8EFC
```

---

141FBEC AA6287C59474E6BC05D99B2964FA090C3A2233BA186515BE7ED1F612970CEE2D  
7AFB81BDD762170481CD0069127D5B05AA993B4EA988D8FDDC186FFB7DC90A6C08F4DF  
435C93402849236C3FAB4D27C7026C1D4DCB2602646DEC9751E763DBA37BDF8FF9406AD  
9E530EE5DB382F413001AEB06A53ED9027D831179727B0865A8918DA3EDBEBCF9B14ED4  
4CE6CBACED4BB1BDB7F1447E6CC254B332051512BD7AF426FB8F401378CD2BF5983CA0  
1C64B92ECF032EA15D1721D03F482D7CE6E74FEF6D55E702F46980C82B5A84031900B1C9  
E59E7C97FBEC7E8F323A97A7E36CC88BE0F1D45B7FF585AC54BD407B22B4154AACC8F6  
D7EBF48E1D814CC5ED20F8037E0A79715EEF29BE32806A1D58BB7C5DA76F550AA3D8A1  
FBFF0EB19CCB1A313D55CDA56C9EC2EF29632387FE8D76E3C0468043E8F663F4860EE12B  
F2D5B0B7474D6E694F91E6DBE115974A3926F12FEE5E438777CB6A932DF8CD8BEC4D073  
B931BA3BC832B68D9DD300741FA7BF8AFC47ED2576F6936BA424663AAB639C5AE4F5683  
423B4742BF1C978238F16CBE39D652DE3FDB8BEFC848AD922222E04A4037C0713EB57A81  
A23F0C73473FC646CEA306B4BCBC8862F8385DDFA9D4B7FA2C087E879683303ED5BDD3  
A062B3CF5B3A278A66D2A13F83F44F82DDF310EE074AB6A364597E899A0255DC164F31C  
C50846851DF9AB48195DED7EA1B1D510BD7EE74D73FAF36BC31ECFA268359046F4EB879  
F924009438B481C6CD7889A002ED5EE382BC9190DA6FC026E479558E4475677E9AA9E3050  
E2765694DFC81F56E880B96E7160C980DD98EDD3DFFFFFFFFFFFFFFFFF. This prime  
number is of length 600, which is much bigger and safe against NFS attacks. It can incorporate  
many users, i.e. a big number of users could be assigned a unique key less than this prime number.  
We also found out that one of the primitive roots of this prime number is 2, thus we hard coded this  
and used it in carrying out all the above functions.

## 6.2.2 Encryption (thrain/src/stand-alone-application/ENCDEC.py)

This section discusses about the implementation of the encryption algorithm using for securing the text. We used the PyCrypto library which already had the implementation of the AES algorithm. The snippet is already described in the section 5.3 of the report. AES uses base 64 encoding and encodes the text in UTF-8 format. It adds some extra bits used as padding bits to make the text more secure.

## 6.2.3 GUI (thrain/src/stand-alone-applicaiton/main.py)

GUI improves user's ability to runt the application. In our application thrain, we used tkinter to build the GUI. This application provided the platform to encrypt, decrypt, upload and download the encrypted files from the online directory. It even contains a link which guides the user on how to use the application.

**Note:** main.py must only be run on the terminal.

## 6.3 Web-application

The web application of the project was used for secure storage on the cloud. The major steps included in the web application are as follows:

## Thrain

[file-upload](#) [file-directory](#) [download-public-key](#) [register-user](#)

The user gets to choose from one of the following options:

- He gets to upload the file to the cloud in a secured manner.
- He gets to choose from the list of all the different files that are present on the cloud provided he has the required credentials for decrypting the file.
- He can also download public key associated with the user with whom the file transfer is to take place.
- To upload his file to cloud storage the user needs to be registered so that his file can be shared with the other desired person in an instant.
- The first step for the user was to register on our platform. On registering the user would be given an automatically generated private key that would be used by the user for transactions. For security purposes the private key of the user is not stored in the database.

## Thrain

Please select the file you wish to upload to the cloud. Make sure the file has been encrypted using the standalone application.

No file selected.

- On selecting the upload file option the user is taken to another page that allows the user to submit the encrypted file.

### **The following are the different files stored on the cloud:**

Username	Uploader
<a href="#">Click here to download public key</a>	satyamsri8
<a href="#">Click here to download public key</a>	parthendo
<a href="#">Click here to download public key</a>	parth
<a href="#">Click here to download public key</a>	tiwari
<a href="#">Click here to download public key</a>	trehan
<a href="#">Click here to download public key</a>	pranjul

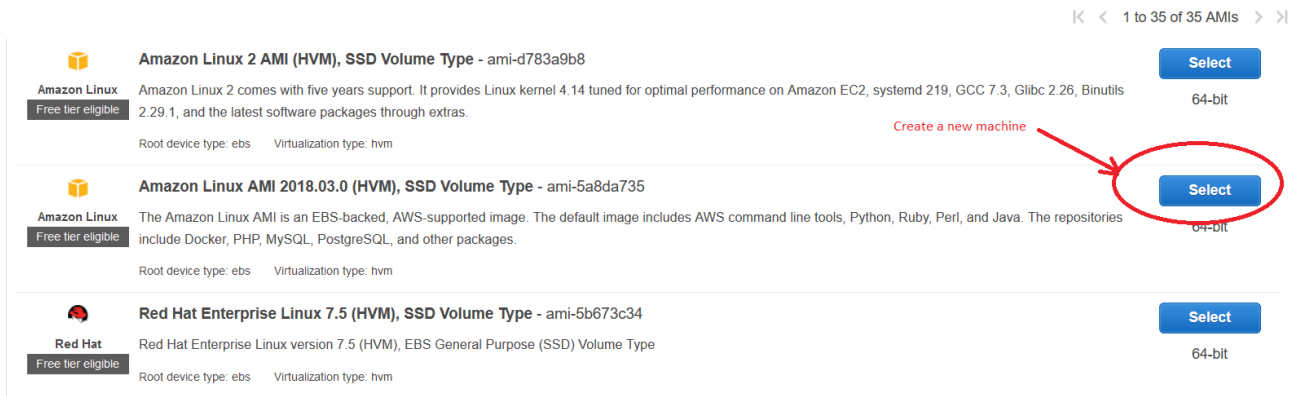
- Clicking the file directory option on the page takes the user to the above page which displays all the different files stored on the cloud.

## Sixth Semester Project Report

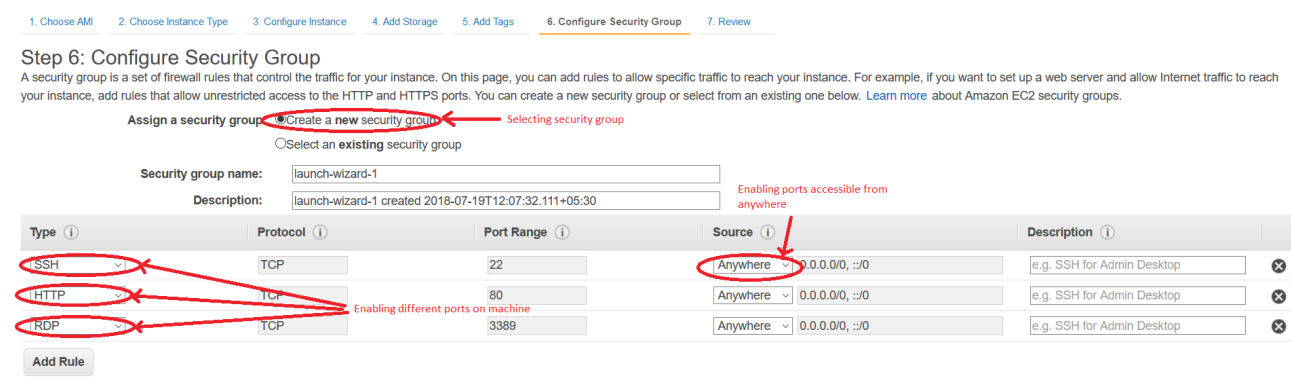
- Selecting different options like download-public key lists the public keys of the registered users which the user can download and use for authenticating the user when someone tries to open his uploaded file.
- Clicking on register user tab take the user to a page where the user registers himself with the platform.

## 6.4 Hosting on Amazon AWS-EC2

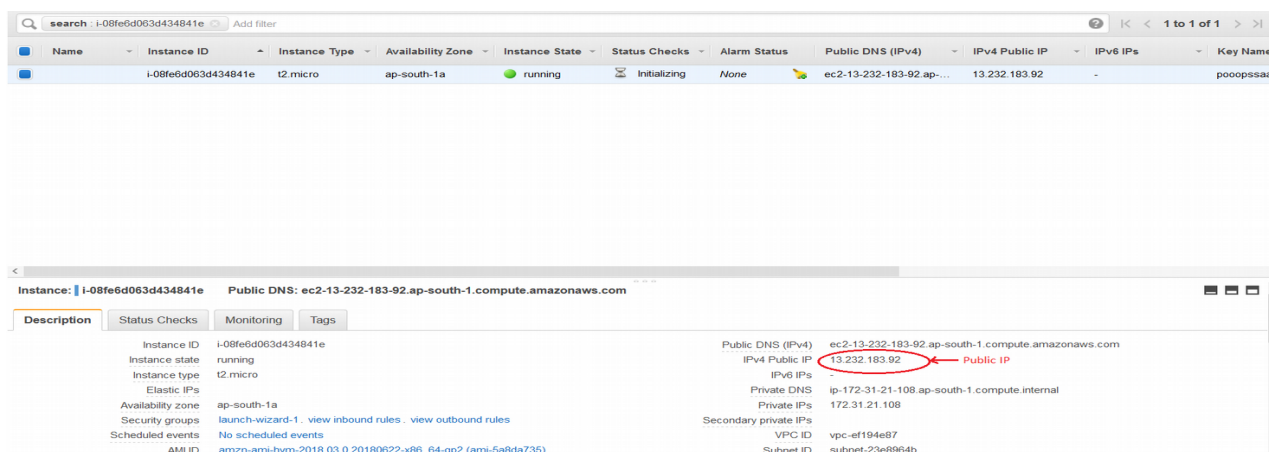
The following steps are to be performed to host the machine on AWS cloud.



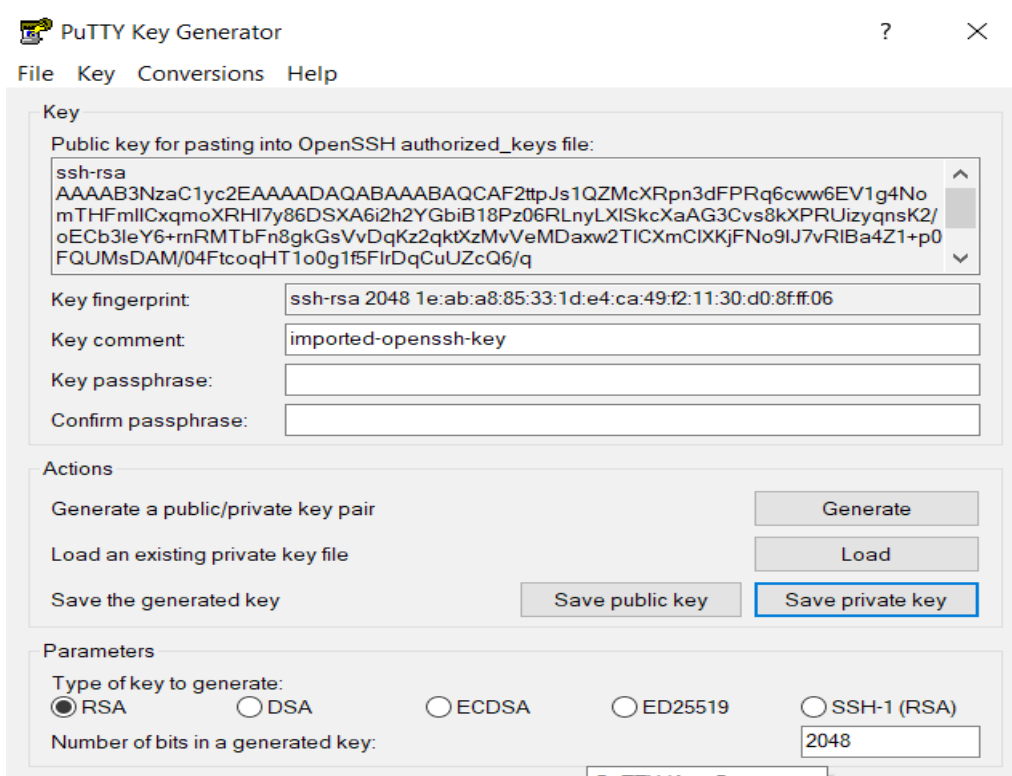
1. First log in to the AWS account and go to EC2 console and select a machine to deploy.



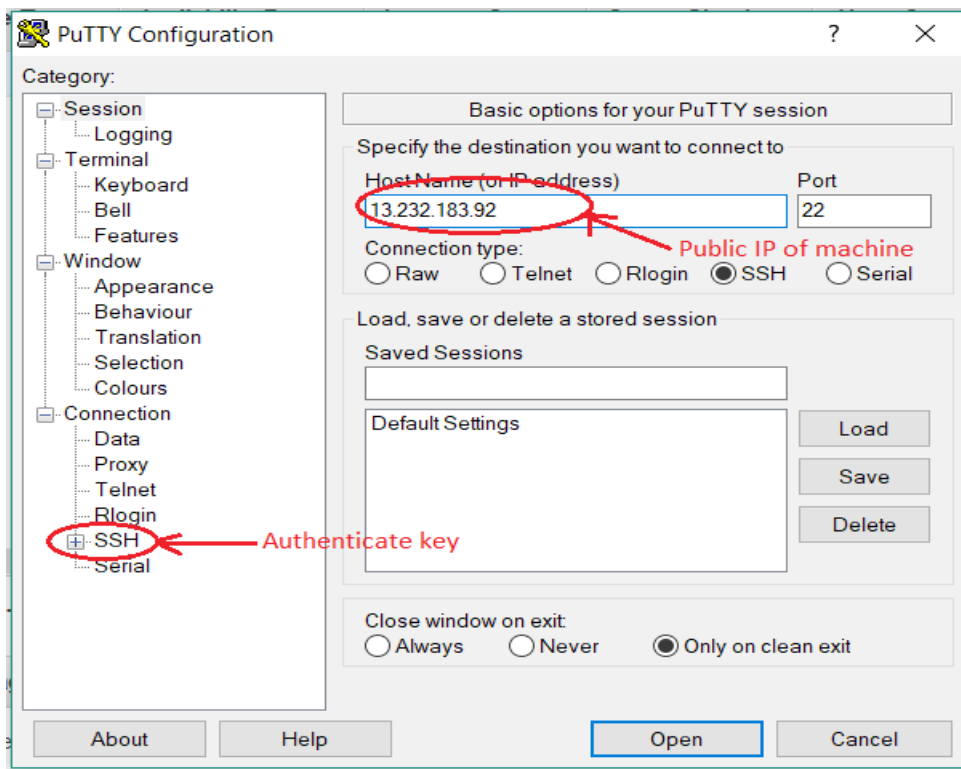
2. Then configure the machines ports as displayed above.



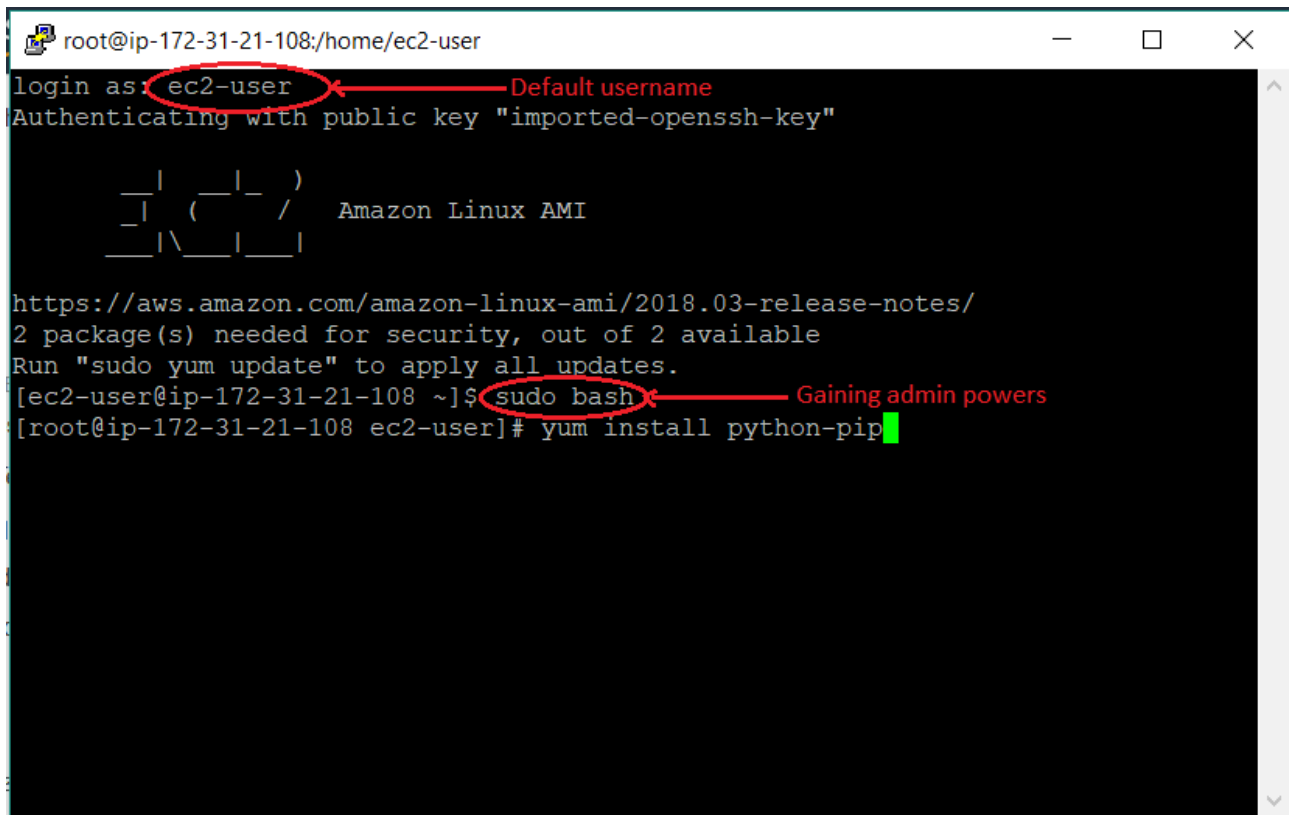
3. When the machine is up and running copy the public IP address of the machine that will be used further.



4. Now open the Putty Key Generator and used the downloaded file to produce private key to be used for connection. Save the private key file.



5. Copy and paste the public IP address in Putty Configuration software and then go to SSH and give the path of the private key file and then click open. This will open the terminal of the remote machine. Once the terminal is open log into the machine.



```
root@ip-172-31-21-108:/home/ec2-user
login as: ec2-user
Authenticating with public key "imported-openssh-key"

  _ | _ | _ )
  _ | ( _ | /   Amazon Linux AMI
  _ | \ _ | _ |

https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
2 package(s) needed for security, out of 2 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-21-108 ~]$ sudo bash
[root@ip-172-31-21-108 ec2-user]# yum install python-pip
```

6. Once you are logged in then install the dependencies like pip, flask, and then host the application on the remote machine. Once the application is running use the public Ip address of the machine to access the application from anywhere.

## 7.0 Conclusion

The proposed project aims to address the problem of secure file storage on the cloud. This method is a basic implementation of the proposed methodology that can be improvised and customised according to the needs. It proposes to use encryption and Diffie Hellman to provide double layer of security to the files that are stored on the cloud.

## References

1. [Diffie, W.; Hellman, M. \(1976\). "New directions in cryptography" \(PDF\). \*IEEE Transactions on Information Theory\*. 22 \(6\): 644–654. doi:10.1109/TIT.1976.1055638. Archived \(PDF\) from the original on 2014-11-29.](#)
2. [Wikipedia](#)
3. Kuhlman, Dave. *"A Python Book: Beginning Python, Advanced Python, and Python Exercises"*. Archived from [the original](#) on 23 June 2012.
4. *"About Python"*. Python Software Foundation. Retrieved 24 April 2012., second section "Fans of Python use the phrase "batteries included" to describe the standard library, which covers everything from asynchronous processing to zip files."
5. [Tkinter Wiki](#)
6. [Flask Wiki](#)
7. [thrain GitHub repository](#)