

**SECURE FILE STORAGE ON CLOUD USING ENCRYPTION AND
DIFFIE HELLMAN**

An End Term Report

**BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING**

SUBMITTED BY

B R REHA (R134217046)

GAURAV RANA (R134217058)

Under the guidance of

MS. TRIPTI MISRA

(Assistant professor (SS), Department of Systemics, SOCS)



**Department of Systemics
SCHOOL OF COMPUTER SCIENCE
UNIVERSITY OF PETROLEUM & ENERGY STUDIES
Bidholi Campus, Energy Acres, Dehradun – 248007.**

DEC - 2020

CANDIDATE’S DECLARATION

We hereby certify that the project work entitled “**SECURE FILE STORAGE ON CLOUD USING ENCRYPTION AND DIFFIE HELLMAN**” in partial fulfillment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with specialization in Cyber Security and Forensics and submitted to the School of Computer Science, Department of Systemics, University of Petroleum & Energy Studies, Dehradun, is an authentic record of our work carried out during a period from **August, 2020 to December, 2020** under the **supervision of Ms. Tripti Misra, Assistant Professor(SS), Department of Systemics.**

The matter presented in this project has not been submitted by us for the award of any other degree of this or any other University.

(B R Reha – R134217046, Gaurav Rana- R134217058)

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Date: 27th Dec, 2020.

Ms. Tripti Misra

(Project Guide)

Dr. Neelu Jyoti Ahuja

HOD- Systemics

School of Computer Science

University of Petroleum &

Energy Studies Dehradun-

248001, UK

ACKNOWLEDGEMENT

We wish to express our deep gratitude to our guide **Ms. Tripti Misra** for all advice, encouragement and constant support she has given us throughout our project work. This project would not have been possible without his support and valuable suggestions.

We sincerely thank our respected Program Head of the Department, **Dr. Neelu Jyoti Ahuja**, for her great support in doing our project.

We would like to thank all our friends for their help and constructive criticism during our project work. Finally, we have no words to express our sincere gratitude to our parents who have shown us this world and for every support they have given us.

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.

TABLE OF CONTENTS

S.No.	Title	Page No.
1	Introduction	6
2	Literature Review	7
3	Problem Statement	9
4	Objectives	9
5	Methodology	10
6	Diagrams	16
7	Requirements	18
8	Implementation	19
9	Conclusion	27
10	Pert Chart	28
	References	29

LIST OF FIGURES

Fig. No.	Title	Page No.
1	Illustration of idea behind Diffie-Hellman	10
2	Schematic of AES structure	13
3	GUI built using Python- Tkinter	14
4	UML Diagram	16
5	Flow chart	17
6	Schedule	28

1. 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 unauthorized person.

2. LITRATURE REVIEW

- In [1] research paper, Key-exchange protocols have been overlooked as a possible means for implementing oblivious transfer (OT). Since, Diffie-Hellman scheme is widely used, our protocol may provide a useful alternative to the conventional methods for implementation of oblivious transfer and a useful primitive in building larger cryptographic schemes.
- In [2] research paper, cryptography key exchange is a strategy by which cryptographic keys are exchanged between two gatherings and those keys are utilized as a part of some cryptographic algorithms like AES. Utilizing those keys sender and recipient exchange encrypted messages. Public key cryptography gives a secured strategy to exchange secret keys. The key exchange issue is the means by which gatherings exchange the keys or data in a communication channel so that nobody else other than sender and recipient can get those. This paper presents Diffie- Hellman key exchange, a procedure which is one of the first public key cryptographic protocols used to build up a secret key between two gatherings over a frail channel. The protocol itself is constrained to exchange of the keys i.e, we are not sharing data while the key exchange, we are making a key together. We start with implementation of algorithm i.e, by building up a mutual secret between two gatherings that can be utilized for secret communication for exchanging information over a public channel. Having no entity authentication mechanism, protocol is effectively assaulted by the man-in-the-middle attack and impersonation attack in practically speaking. Diffie-Hellman is appropriate for utilization in information communication however is less frequently utilized for information storage or archived over long period of time.
- In [3] research paper, two kinds of contemporary developments in cryptography are examined. Widening applications of teleprocessing have given rise to a need for new types of cryptographic systems, which minimize the need for secure key distribution channels and supply the equivalent of a written signature. This paper suggests ways to solve these currently open problems. It also discusses how the theories of communication and computation are beginning to provide the tools to solve cryptographic problems of long standing.
- In [4] research paper, Oblivious Transfer (OT) is the fundamental building block of cryptographic protocols. In this paper we describe the simplest and most efficient protocol for 1- out-of-n OT to date, which is obtained by tweaking the Diffie-Hellman

key-exchange protocol. The protocol achieves UC-security against active and adaptive corruptions in the random oracle model. Due to its simplicity, the protocol is extremely efficient and it allows to perform m 1-out-of- n OTs using only:

Computation: $(n+1)m+2$ exponentiations (mn for the receiver, $mn+2$ for the sender) and Communication: $32(m+1)$ bytes (for the group elements), and $2mn$ ciphertexts.

We also report on an implementation of the protocol using elliptic curves, and on a number of mechanisms we employ to ensure that our software is secure against active attacks too. Experimental results show that our protocol (thanks to both algorithmic and implementation optimizations) is at least one order of magnitude faster than previous work.

- In [5] research paper, 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.
- In [6] research paper, Security is often cited as one of the most contentious issues in Cloud computing. It is argued that as the Cloud is intended to handle large amounts of data, attackers can be sure of a high pay-off for their activities. In addition, to benefit from the Economies of scale, the applications and operating systems are homogenized to a few images restricting the variations of products used within the Cloud. Millions of users are surfing the Cloud for various purposes; therefore, they need highly safe and persistent services. The future of cloud, especially in expanding the range of applications, involves a much deeper degree of privacy, and authentication. We propose a simple data protection model where data is encrypted using AES and Authenticated by Diffie Hellman algorithm before it is launched in the cloud, thus ensuring data confidentiality and security.

3. PROBLEM STATEMENT

We needed a system in which two users, without knowing the secret key, can independently generate same key at both ends that could later be used to encrypt and decrypt the text.

4. OBJECTIVE

Developing an application that provides a secure file storage on the cloud.

a) Sub Objectives:

- Encryption of the file
- User authentication
- Key for AES encryption
- Relationship between this key and Diffie Hellman Key exchange protocol.

5. METHODOLOGY

1. Diffie Hellman Key Exchange

The Diffie-Hellman key exchange was the primary widely used method of safely developing and exchanging keys over an insecure channel. DH is one among the earliest practical samples of public key exchange implemented within the sector of cryptography.

To implement Diffie-Hellman, the two end users Alice and Bob, while communicating over a channel they know to be private, mutually agree on positive whole numbers p and q , such that p may be a prime and q may be a generator of p . The generator q may be a number that, when raised to positive whole-number powers but p , never produces an equivalent result for any two such whole numbers. The value of p could also be large but the worth of q is typically small.

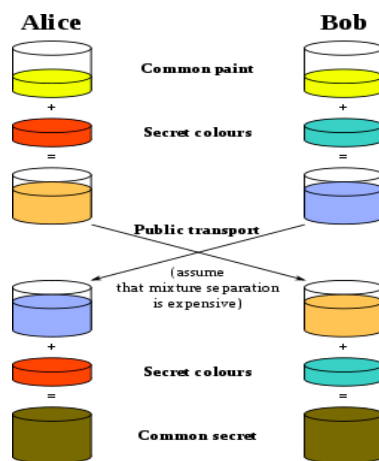


Figure 1: Illustration of idea behind Diffie-Hellman

Diffie–Hellman key exchange establishes a shared secret between two parties which will 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 colors instead of very large numbers.

Once Alice and Bob have agreed on p and q privately, they choose positive whole-number personal keys a and b , both but the prime-number modulus p . Neither user divulges their personal key to anyone; ideally, they memorize these numbers and don't write them down or store them anywhere.

The most serious limitation of Diffie-Hellman in its basic or "pure" form is that the lack of authentication. Communications using Diffie-Hellman all by itself are susceptible to man within the middle attacks. Ideally, Diffie-Hellman should be utilized in conjunction with a recognized authentication method like digital signatures to verify the identities of the users over the general public communications medium. Diffie-Hellman is compatible to be used in digital communication but is a smaller amount often used for data stored or archived over long periods of your time.

1.1. 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).

1.2. 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. $(\text{public key of Bob}^{\text{Private Key of Alice}}) \bmod p$

$\Rightarrow (10^4) \% 23 \Rightarrow 10000 \% 23 \Rightarrow 18$

7. On the other side, Bob also uses a similar method to generate a secret

key, i.e. $(\text{public key of Alice}^{\text{Private Key of Bob}}) \bmod p$

$\Rightarrow (4^3) \% 23 \Rightarrow 64 \% 23 \Rightarrow 18$

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.

2. 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.

2.1. 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 times 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 designs

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.

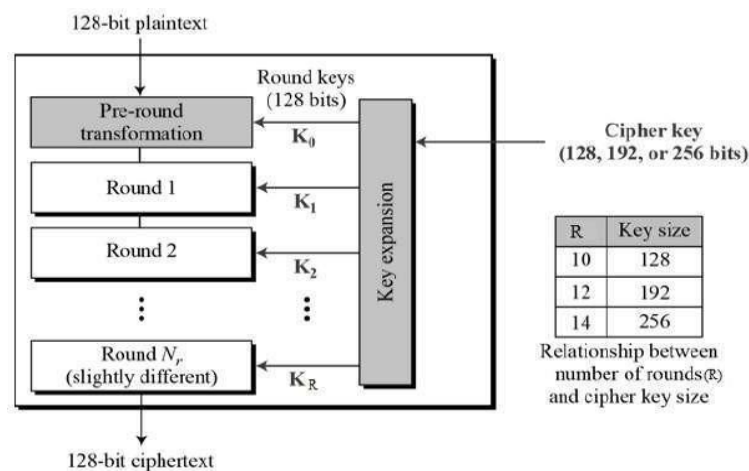


Figure 2: The schematic of AES structure

3. 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. [3] 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.

3.1. 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. [5] 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 BeSec, the framework of the hour.

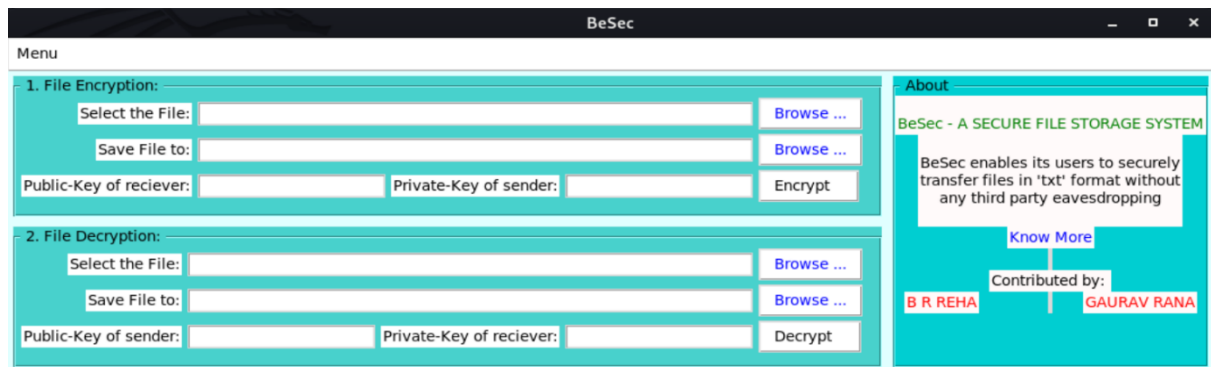


Figure 3: GUI built using python-tkinter

3.2. 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.

3.3. 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

4. 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.

5. 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. DIAGRAMS

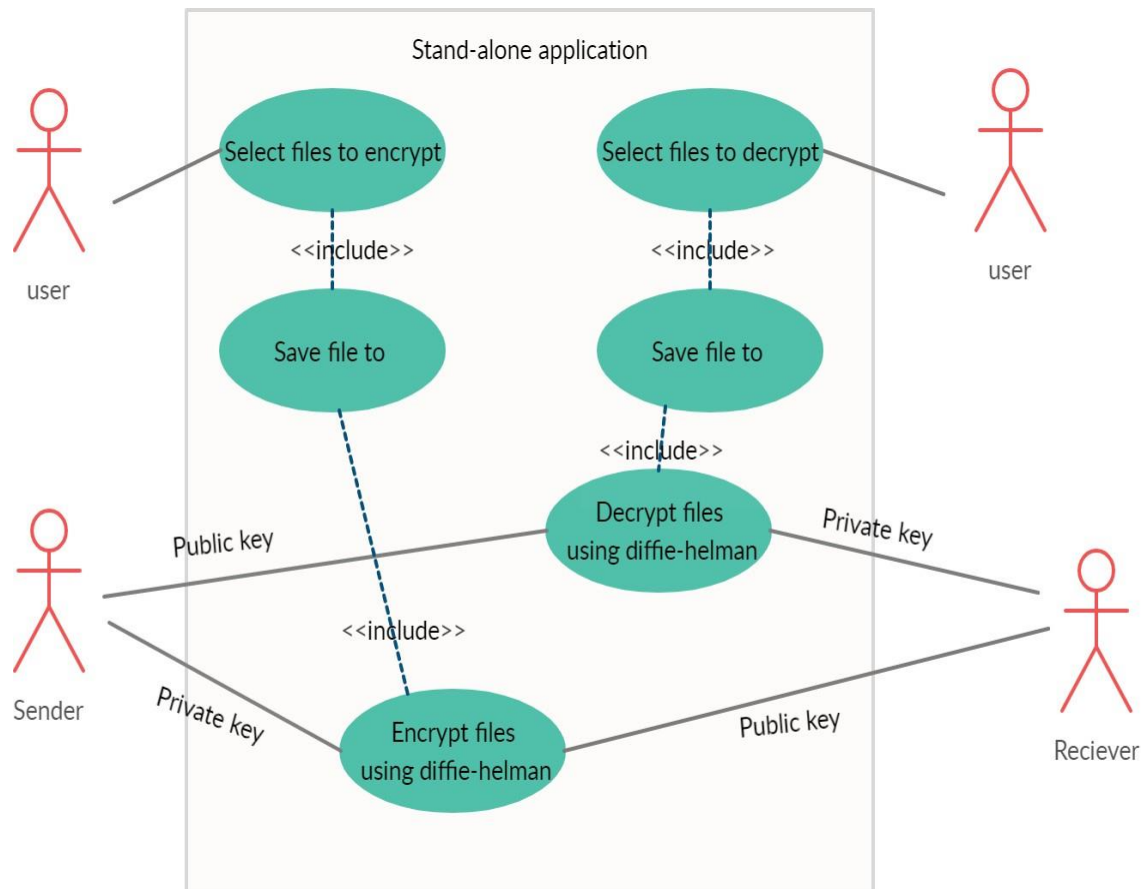


Figure 4: UML Diagram

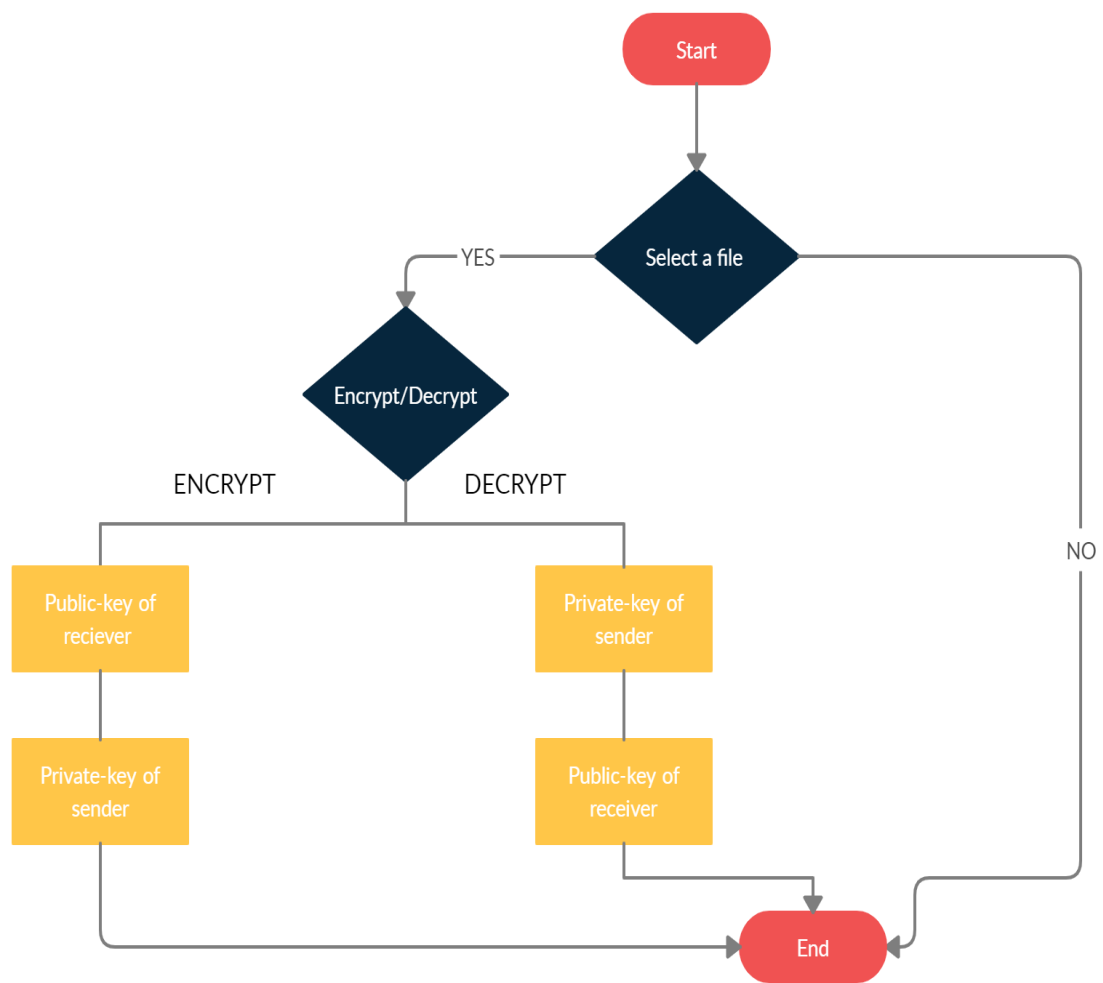


Figure 5: Flow Chart

7. REQUIREMENTS

7.1. Hardware requirements:

One personal computer with:

- 7.1.1. Minimum 4 gigabytes of RAM
- 7.1.2. Processor- Core i3
- 7.1.3. 100 GB Hard Disk
- 7.1.4. Internet Connection

7.2. Software requirements:

- 7.2.1. Operating System- Windows 8 (Client), Ubuntu 10.10 (Server)
- 7.2.2. Application- Python 2.7 and Tkinter 8.5
- 7.2.3. Cloud Platform- AWS
- 7.2.4. Front End- HTML

8. IMPLEMENTATION

This section describes the methodologies and implementation details of the targeted framework. The contents in this section are organized as follows: Section 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 2 describes the details of the stand-alone application and the details how we build the application from scratch.

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 2768 could be computed. This was approximately 232 digits. Thus, we concluded that a larger prime number is needed in the process. Therefore, we used a prime number having 600 digits.

While analyzing 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.

2. Stand-alone-application (BeSec /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.

2.1. Diffie-Hellman (BeSec /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

```
0xFFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC740
20BBEA63B139B22514A08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1
356D6D51C245E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE386BF
B5A899FA5AE9F24117C4B1FE649286651ECE45B3DC2007CB8A163BF0598DA48361C
55D39A69163FA8FD24CF5F83655D23DCA3AD961C62F356208552BB9ED52907709696
6D670C354E4ABC9804F1746C08CA18217C32905E462E36CE3BE39E772C180E86039B
2783A2EC07A28FB5C55DF06F4C52C9DE2BCBF6955817183995497CEA956AE515D22
61898FA051015728E5A8AAAC42DAD33170D04507A33A85521ABDF1CBA64ECFB850
458DBEF0A8AEA71575D060C7DB3970F85A6E1E4C7ABF5AE8CDB0933D71E8C94E0
4A25619DCEE3D2261AD2EE6BF12FFA06D98A0864D87602733EC86A64521F2B18177
B200CBBE117577A615D6C770988C0BAD946E208E24FA074E5AB3143DB5BFCE0FD1
08E4B82D120A92108011A723C12A787E6D788719A10BDBA5B2699C327186AF4E23C1
A946834B6150BDA2583E9CA2AD44CE8DBBBC2DB04DE8EF92E8EF141FBECAA628
7C59474E6BC05D99B2964FA090C3A2233BA186515BE7ED1F612970CEE2D7AFB81B
DD762170481CD0069127D5B05AA993B4EA988D8FDDC186FFB7DC90A6C08F4DF
435C93402849236C3FAB4D27C7026C1D4DCB2602646DEC9751E763DBA37BDF8FF94
06AD9E530EE5DB382F413001AEB06A53ED9027D831179727B0865A8918DA3EDBEB
CF9B14ED44CE6CBACED4BB1BDB7F1447E6CC254B332051512BD7AF426FB8F40138
CD2BF5983CA01C64B92ECF032EA15D1721D03F482D7CE6E74FEF6D55E702F46980C
82B5A84031900B1C9E59E7C97FBEC7E8F323A97A7E36CC88BE0F1D45B7FF585AC54
```

BD407B22B4154AACC8F6D7EBF48E1D814CC5ED20F8037E0A79715EEF29BE32806A
1D58BB7C5DA76F550AA3D8A1FBFF0EB19CCB1A313D55CDA56C9EC2EF29632387F
E8D76E3C0468043E8F663F4860EE12BF2D5B0B7474D6E694F91E6DBE115974A3926F
12FEE5E438777CB6A932DF8CD8BEC4D073B931BA3BC832B68D9DD300741FA7BF8
AFC47ED2576F6936BA424663AAB639C5AE4F5683423B4742BF1C978238F16CBE39D
652DE3FDB8BEFC848AD922222E04A4037C0713EB57A81A23F0C73473FC646CEA306
B4BCBC8862F8385DDFA9D4B7FA2C087E879683303ED5BDD3A062B3CF5B3A278A6
6D2A13F83F44F82DDF310EE074AB6A364597E899A0255DC164F31CC50846851DF9A
B48195DED7EA1B1D510BD7EE74D73FAF36BC31ECFA268359046F4EB879F92400943
8B481C6CD7889A002ED5EE382BC9190DA6FC026E479558E4475677E9AA9E3050E276
5694DFC81F56E880B96E7160C980DD98EDD3DFFFFFFFFFFFFFFFFFFFFF.

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.

2.2. Encryption (BeSec /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.

2.3. GUI (BeSec /src/stand-alone-application/main.py)

GUI improves user's ability to run the application. In our application BeSec, 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.

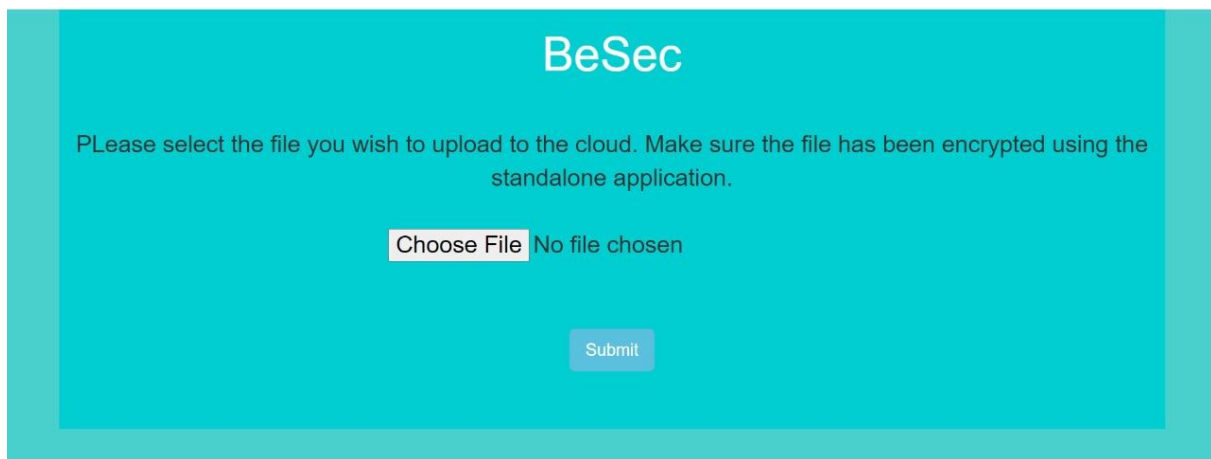
Note: main.py must only be run on the terminal.

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:



- 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.



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

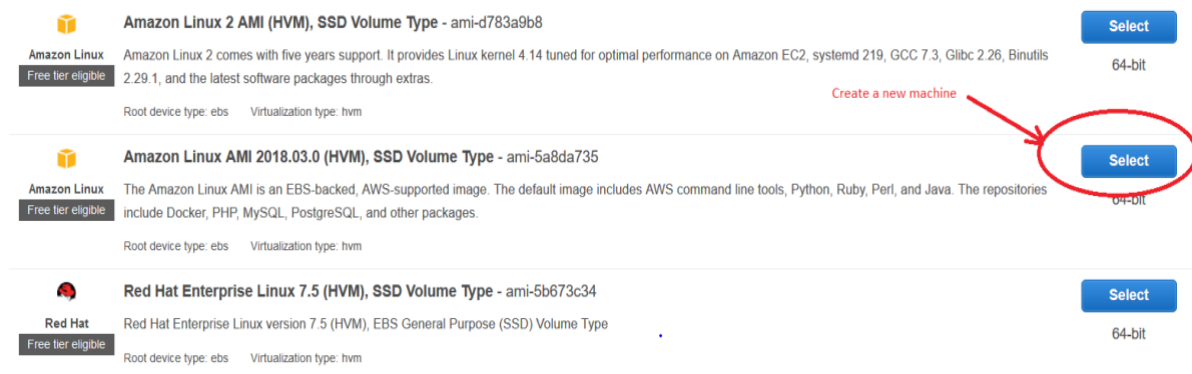
The following are the different files stored on the cloud:

Username	Uploader
Click here to download public key	satyamsri8
Click here to download public key	parthendo
Click here to download public key	parth
Click here to download public key	tiwari
Click here to download public key	trehan
Click here to download public key	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.
- 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.

4. Hosting on Amazon AWS-EC2

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

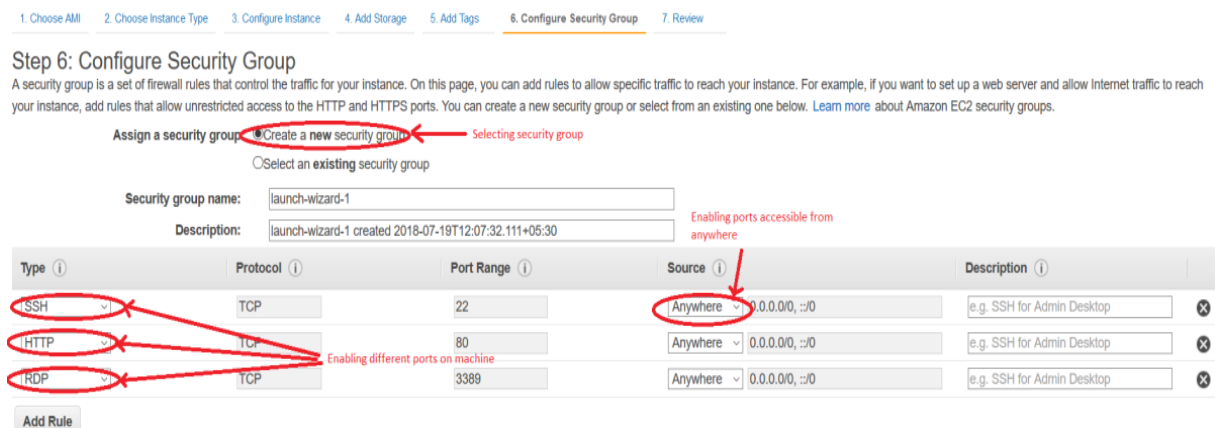


Amazon Linux 2 AMI (HVM), SSD Volume Type - ami-d783a9b8
 Amazon Linux 2 comes with five years support. It provides Linux kernel 4.14 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras.
 Root device type: ebs Virtualization type: hvm **Select** 64-bit

Amazon Linux AMI 2018.03.0 (HVM), SSD Volume Type - ami-5a8da735
 The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.
 Root device type: ebs Virtualization type: hvm **Select** 64-bit

Red Hat Enterprise Linux 7.5 (HVM), SSD Volume Type - ami-5b673c34
 Red Hat Enterprise Linux version 7.5 (HVM), EBS General Purpose (SSD) Volume Type
 Root device type: ebs Virtualization type: hvm **Select** 64-bit

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



Step 6: Configure Security Group
 A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

Assign a security group: ☒ Create a new security group ☐ Select an existing security group

Security group name: launch-wizard-1
 Description: launch-wizard-1 created 2018-07-19T12:07:32.111+05:30

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	Anywhere	e.g. SSH for Admin Desktop
HTTP	TCP	80	Anywhere	e.g. SSH for Admin Desktop
RDP	TCP	3389	Anywhere	e.g. SSH for Admin Desktop

Add Rule

2. Then configure the machines ports as displayed above.

The screenshot shows the AWS Management Console. The top table lists EC2 instances. The instance 'i-08fe6d063d434841e' is in the 'running' state. Below, the 'Instance: i-08fe6d063d434841e' details are shown. The 'Public DNS (IPv4)' is 'ec2-13-232-183-92.ap-south-1.compute.amazonaws.com'. The 'IPv4 Public IP' is '13.232.183.92', which is circled in red and labeled 'Public IP' with a red arrow. Other details include 'Private DNS', 'Private IPs', 'Secondary private IPs', 'VPC ID', and 'Subnet ID'.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IPv6 IPs	Key Name
	i-08fe6d063d434841e	t2.micro	ap-south-1a	running	Initializing	None	ec2-13-232-183-92.ap-south-1.compute.amazonaws.com	13.232.183.92	-	poopssai

Instance: i-08fe6d063d434841e		Public DNS: ec2-13-232-183-92.ap-south-1.compute.amazonaws.com	
Description		Status Checks	
Instance ID	i-08fe6d063d434841e	Public DNS (IPv4)	ec2-13-232-183-92.ap-south-1.compute.amazonaws.com
Instance state	running	IPv4 Public IP	13.232.183.92
Instance type	t2.micro	IPv6 IPs	-
Elastic IPs		Private DNS	ip-172.31.21.108.ap-south-1.compute.internal
Availability zone	ap-south-1a	Private IPs	172.31.21.108
Security groups	launch-wizard-1, view inbound rules, view outbound rules	Secondary private IPs	
Scheduled events	No scheduled events	VPC ID	vpc-ef194e87
AMI ID	amzn-ami-hvm-2018.03.0.20180622-x86_64-gp2 (ami-5a8da735)	Subnet ID	subnet-23e9964b

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

The screenshot shows the PuTTY Key Generator window. The 'Key' section displays a public key for pasting into an OpenSSH authorized_keys file. The key fingerprint is 'ssh-rsa 2048 1e:ab:a8:85:33:1d:e4:ca:49:f2:11:30:d0:8f:ff:06'. The key comment is 'imported-openssh-key'. The 'Actions' section has buttons for 'Generate', 'Load', 'Save public key', and 'Save private key'. The 'Parameters' section shows 'Type of key to generate' as 'RSA' and 'Number of bits in a generated key' as '2048'.

PuTTY Key Generator

File Key Conversions Help

Key

Public key for pasting into OpenSSH authorized_keys file:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQCAFE2tpJs1QZMcXRpn3dFPRq6cww6EV1g4No
mTHFmIlCxmQXRHl7y86DSXA6i2h2YGbiB18Pz06RLnyLXISkcXaAG3Cvs8kXPRUizyqnsK2/
oECb3leY6+rnRMTbFn8gkGsVvDqKz2qktXzMvVeMDaxw2TicXmCIXKjFNo9IJ7vRIBa4Z1+p0
FQUMsDAM/04FtcoqHT1o0g1f5FlrDqCuUZcQ6/q
```

Key fingerprint: ssh-rsa 2048 1e:ab:a8:85:33:1d:e4:ca:49:f2:11:30:d0:8f:ff:06

Key comment: imported-openssh-key

Key passphrase:

Confirm passphrase:

Actions

Generate a public/private key pair **Generate**

Load an existing private key file **Load**

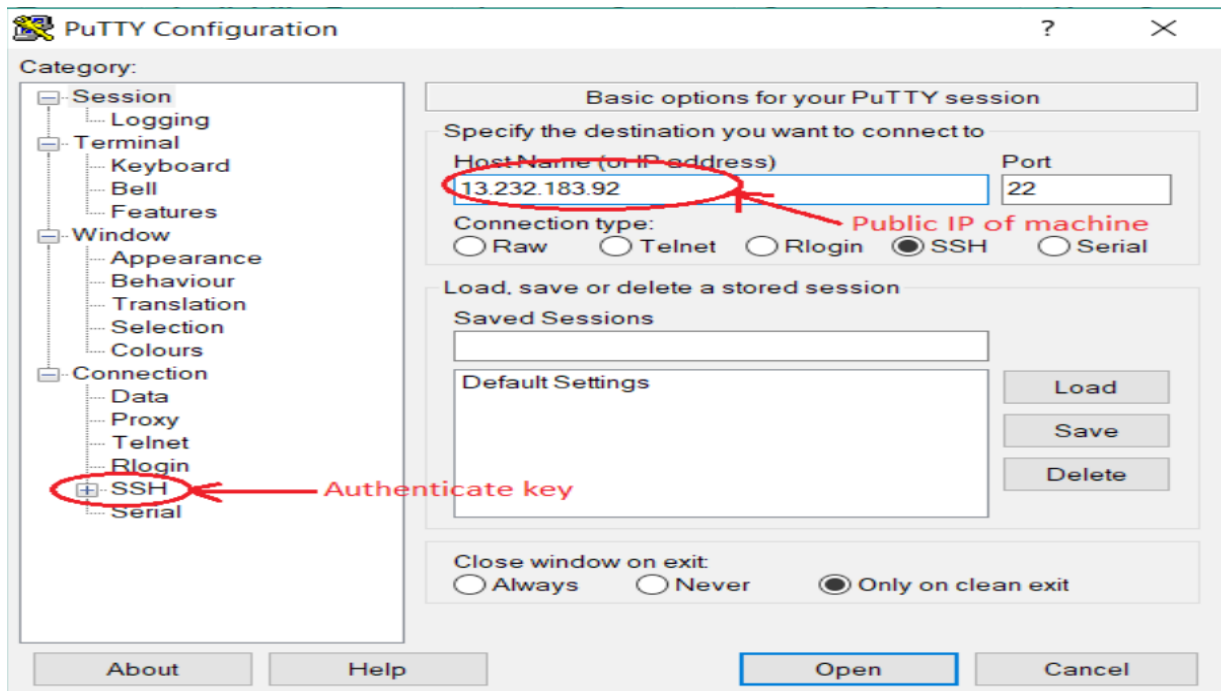
Save the generated key **Save public key** **Save private key**

Parameters

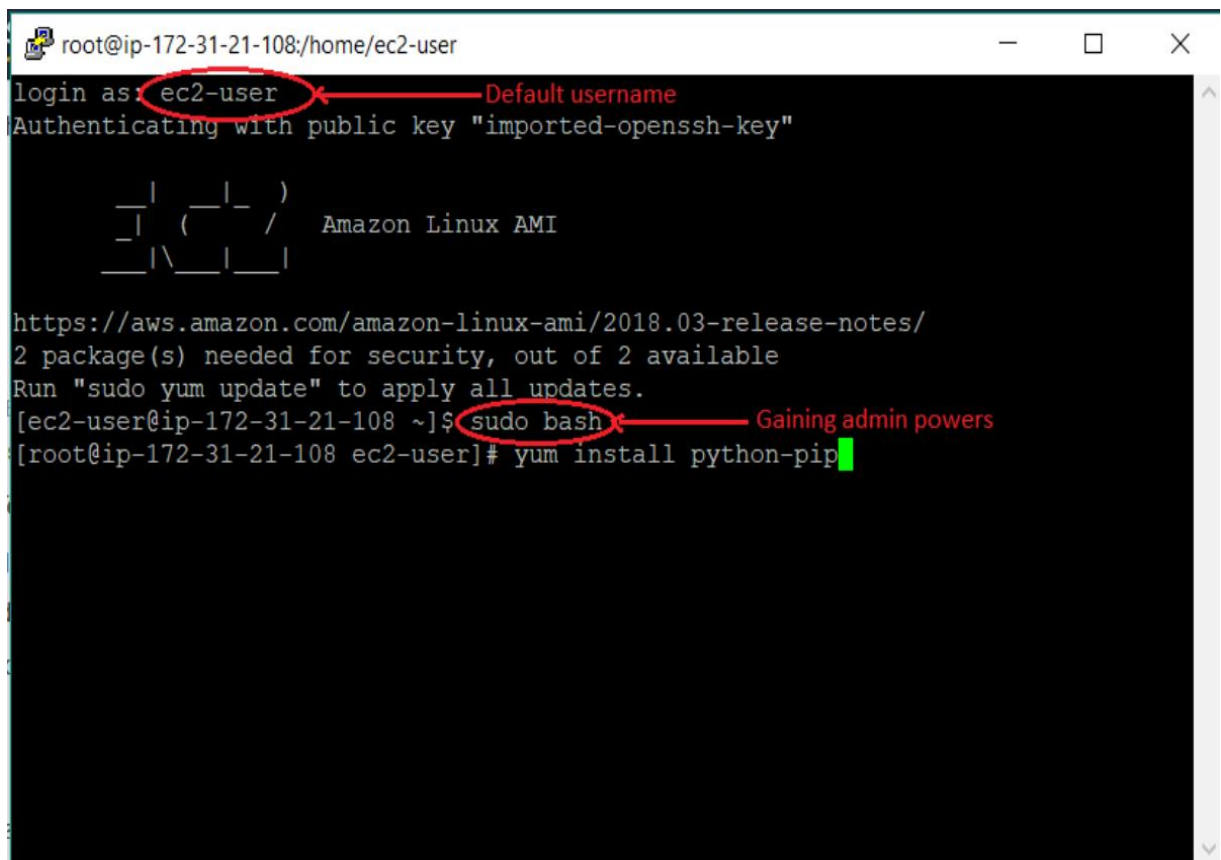
Type of key to generate: ☒ RSA ☐ DSA ☐ ECDSA ☐ ED25519 ☐ SSH-1 (RSA)

Number of bits in a generated key: 2048

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.



- 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.



- 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.

9. 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 customized 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.

10. PERT CHART

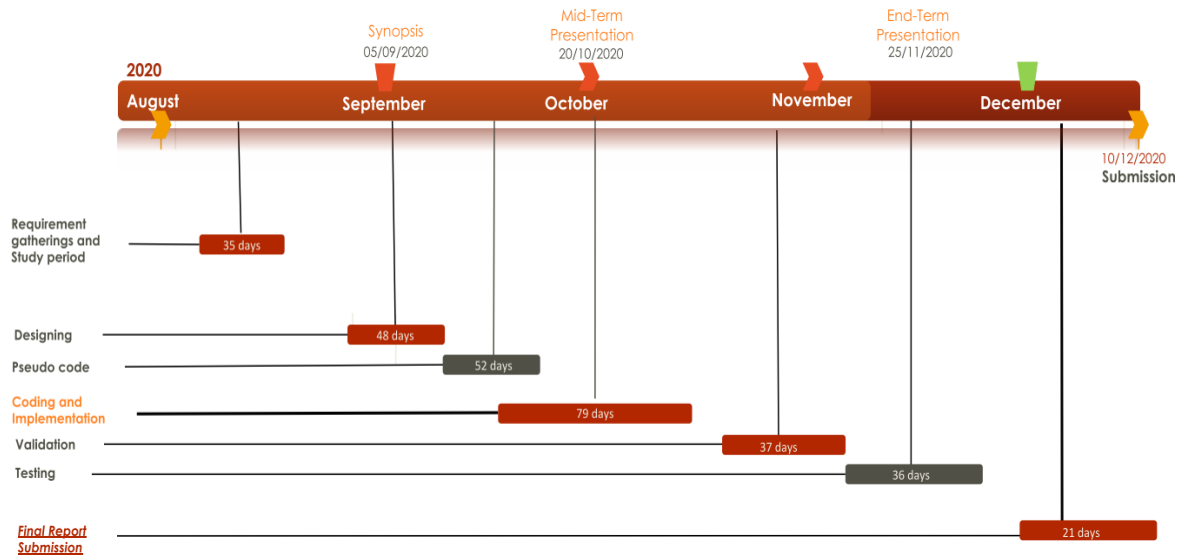


Figure 6: Schedule

REFERENCES

- [1] Abhishek Parakh, 10th January 2008, Oblivious Transfer Based on Key Exchange. Available Online at: [<https://arxiv.org/abs/0705.0178>]
- [2] Sivanagaswathi Kallam, 30th September 2015, Diffie-Hellman: Key Exchange and Public Key Cryptosystem. Available Online at: [<http://cs.indstate.edu/~skallam/doc.pdf>]
- [3] Whitfield Diffie and Martin E. Hellman, 6th November 1976, New Directions in Cryptography. Available Online at: [<https://ee.stanford.edu/~hellman/publications/24.pdf>]
- [4] Tung Chou and Claudio Orlandi, 15th August 2015, The Simplest Protocol for Oblivious Transfer. Available Online at: [<https://eprint.iacr.org/2015/267.pdf>]
- [5] Celeste Murnal and K. Pramilarani, 5th May 2019, Secure Text Transfer. Available Online at: [<http://ijics.com/gallery/58-may-1148.pdf>]
- [6] Rameshwari Malik and Pramod Kumar, 1st May 2015, Cloud Computing Security Improvement using Diffie-Hellman and AES. Available Online at: [<https://research.ijcaonline.org/volume118/number1/pxc3903030.pdf>]

APPROVED BY:

**Ms. Tripti Misra
(Project Mentor)**

**Dr. Neelu Jyoti Ahuja
(HOD- Systemics School of
Computer Science)**