

Chapter 1: Introduction

1.1 Background

A Key Management System (KMS) facilitates a dynamic way to generate, exchange, store and replace keys within a cryptosystem. It plays a potential role to provide security of the cryptographic keys typically between a system or user.

Encryption is combination of several mathematical rules that is usually used to restrict access of data as well as to make it accessible only by an authenticated entity. The sensitive data can be accessible by the intruders if not enough barrier is created. A key management system will fix the issues of security of encryption keys while in rest as well as on transit, secured distribution of keys within a system pair, proper backup and recovery of encryption keys and regenerate and destroy keys depending on the situation demands.

The key problem of a key management is to efficiently generate and secure encryption keys that is required to be executed each time a user attempts to communicate with another system usually by sending a message. A large number of user request to use encryption keys makes the job of managing the keys much complicated. Besides, it is equally important to provide adequate protection to these keys when they are stored in a system.

1.2 Problem Statement and motivation

When multiple users are frequently communicating with each other usually in an organization environment, the creation and exchange of these keys can be quite complex for any system. Besides, it is essential to have a secured storage system to handle the backup and recovery process of these keys. The most difficult problem is to secure the key when an agreement is made between two systems before the exchange process of an encryption key.

Nowadays, the number of cloud users are increasing by no bounds. Hence, cloud providers are required to offer massive data storage capacity in servers with a view to accommodate the vast amount of user's data. As a result, the occurrences of data breaching related attacks can be noticed more. Besides, the defense mechanisms regarding the authentication in the cloud systems are not strong enough and the possibilities of getting cracked is higher. As a resultant of that assailants can attempt to apply some trick to install malicious program inside the cloud server and perform

different type of calamitous things including deleting user's useful data, wielding data which can be cause serious loss to an enterprise or individual.

Furthermore, data availability is a big issue for cloud computing since user's might need to access the data from different geographical locations at different time. It is highly expensive to settle multiple data centers. Besides, only offering of strong encryption mechanism is not sufficient. The management of the keys is equally important as it is simple fact that a user can forget or lost his password. Another complex issue on cloud is trust regarding the control of data. In a centralized cloud platform, the users need to agree to the fact that cloud providers will provide safety to their data when stored on cloud server. The integrity and confidentiality of the data in this matter becomes the major issue since the cloud providers can manipulate the data on server as well as decrypt it.

In large data centers where thousands of transmissions of data are performed continuously, the management of these cryptographic keys becomes more critical since the elevated number of required cryptographic keys and their management within several physical and logical storage media. This management of these cryptographic keys turns out to be more intricate when it comes to cloud environment where tasks are divided among several entities who are responsible for controlling tasks related to networking as well as computing. However, the different actors related to the several control activities in a cloud environment can be categorized into three entities- the vendors, brokers and customers.

However, a key management integrated with strong security mechanisms in terms of data conceal methods, managing encryptions keys and authentication process deployed in a decentralized network can be a good solution to fix most of the issues discussed above. A decentralized cloud will solve the trust issue since the users will no longer need to store data in dedicated server as well as the data availability problem with peer-to-peer network. On the other hand, the combined use of cryptography and encryption can solve the data integrity issue.

1.3 Project Aims and Objectives

The key focus to concentration and aims of the research is to provide a secured user controlled key management system on peer-to-peer distributed cloud network that allows the clients to use a command line tools to encode their sensitive data using hybrid cryptographic techniques that

comprises LSB and AES256 generally known as steganography. The cloud server will have secured authentication system to allow only registered users to use cloud service. Therefore, the server will be capable of securely exchange distribution keys with the clients and these keys will be stored as well as recovered on demand. To achieve the aforementioned aims, the following objectives have been set-

- To develop a key management server with secured login authentication on decentralized cloud server to allow effectively backup and recovery of encryption keys on demand.
- To implement a secured command line tool for client to use combined cryptographic techniques (LSB and SHA256) to encode/decode data in image with user given password to ensure data integrity and confidentiality during transmission.

1.4 Project Scopes

The offered features in the objective section leads to the following scopes-

- Implemented key management system on decentralized cloud supports Trusted Platform Module (TPM) and allows authentication-based login for clients in a secured manner.
- Encrypting the data file using Advanced Encryption Standard (AES256) technique.
- Embedding the processed data file into image using Least Signification Bits (LSB) and encrypt the image with user given password.
- Encoding the password into the same image and transmitting the encryption key to the server.
- Allowing encryption key backup and recovery option and decrypting encoded image to output plain text in text file.

However, the following features are out of the scope of this research-

- Unestablished authentication scheme to perform client identification.
- Uses simple symmetric encryption technique while transmitting encrypted keys.
- Inadequate features on server such as DNS based login, blocking IP, showing clients list.

1.5 Summary

To sum up, the implemented key management system will definitely solve a lot of issues present in existing system. The combined cryptographic techniques used in the system makes it almost impossible to crack plain text from the image. The use of encryption of plain text before embedding reforms the data order in file that looks like blocks of random characters. Thus, even if someone detects the presence of info using steganalysis tools, they can see only random characters and can never extract the original plain text. Nevertheless, the encoded image file can be stored safely anywhere as well as transferred via email or USB storage.

Chapter 2: Literature Review

2.1 Introduction

In this chapter, the existing key management system deployed on cloud environment and several types of security measures implemented on them have been discussed in detail. The first part of the current chapter illustrates different types of key management system adopted with different types of security measures. The proposed KMS on a decentralized cloud can be more effective and scalable. In addition, the following parts in this chapter projects different types of cloud structures and cryptographic measures are generally used on current systems available commercially.

2.2 Existing Key Management System on Cloud Environment

A key management plays an important role to enhance security of data specially during their transmission session. The most popular and primary technique to secure any data is provided by several encryption methods. By having access to the key, anybody can see the hidden information usually inside a message. Thus, it is mandatory to provide enough security measures for these keys to ensure data confidentiality and integrity. The primary task of a Key Management System (KMS) is to create, negotiate and store in a database with secured mechanism deployed. KMS also prevents encryption keys from getting lost, corrupted as well as facilitates protection from unauthorized access. There are generally two types of key management that are popular and used by many enterprises and individuals- Remote Key Management Service (RKMS) and Client-Side Key Management Service (CSKM). Both structures consist of many advantages and disadvantages.

In RKMS, encryption key (EK) is established and distributed from a remote server where the control is given to the cloud provider and the server will remotely encrypt/decrypt data as needed. On the other hand, CSKM is based on decentralized structure of cloud where the control is given to the users. The clients usually deal with the encryption/decryption and store it on their personal computer. Whereas, the cloud providers are unable to perform decryption of the encrypted data as they don't have encryption key. However, they will provide the necessary storage required to keep the encrypted data.

(Yoo SM, Park PK, Shin JS, Oh JS, Ryu HY, Ryou Jc, *et al.*, 2013) has researched on user-based control scheme in Institute of Electrical and Electronics Engineers that effectively ensured integrity of their data from the Cloud Service Providers (CSV). However, another fabulous design was proposed by (Fakhar F, Shibli MA.,2013) that can be used to manage private keys with more competent way by integrating Shamir's Enhanced method. According to the scheme, the entire key is broken into several pieces. Considering the total number of encryption key as X pieces are stored on Y number of servers where each server will store only piece of the EK.

EK1, EK2, EK3, EK4, EK5...EKX

X=Y

Therefore, the algorithm looks as shown below:

$$\sum_{x=1}^{y=x-1} Y = X - 1$$

Figure 1: Shamir's Algorithm

In case, the encryption key to break does not meet the minimum length needed, a series of special type characters are generated to be assigned. Thus, the number X is actually dependent on available free size on disks/server.

Moreover, a more effective and scalable technique to manage huge data on servers have been introduced by (Wang Y, Li Z, Sun Y,2015) where a model of the system can only be designed if communication takes place directly between the server and cryptographic key management system layer that provides secure user authentication as well as proper management of EK.

In addition, (Damgård I, Jakobsen TP, Nielsen JB, Pagter JI, 2013) has suggested an attractive scheme for verifying key Publicly by applying Robert's scheme of sharing secret. In this algorithm, having acknowledgement of global key and (n-1) portion are not adequate to fetch shared decrypted form of the cipher text.

A research has been conducted by Aneesh Singh Shekhawat on RSA set of rules, every piece of key is in elements such that it can't be manipulated without the personal or confidential access key owned by the proprietor itself, and the encryption can't be done if its exceeds the edge value (t).

According to the research done by Wang et al., rekeying strategy is used in which information will again be enciphered after the operators makes use of the information such that it may be extra secured at the same time as of records and simultaneously safeguards their personal information as well. Besides, from deliberation of Chen and Yang, exile foundation mechanism is discussed in which multiple operators are indulged in sharing the records and at the same time, encryption is mandatory on every message by a justifiable procedure which sanctions them to confine to a subset of operators taking part simultaneously. however, this method has a drawback of concordance. In proxy re-encryption schemes, cipher text is shared with the use of one confidential key, wherein a intermediary encrypt the message for all the other party such that they are able to decrypt the cipher textual content.

In attribute-based encryption(ABE), index is associated with a set of attributes and that information can be easily deciphered in the event which satisfy the requirements of the admission commands connected to the confidential key of the proprietor. Alongside this, according to the observation of Jia et al., using multiple group key management (GKM), the information among the associates of the secured foundation is kept confidential and delegating them with a few protocols. In addition, from the analysis of Nabeel et al., with composite encoding, we will be able to encrypt our information using ABE or online network for the use of the secret key with re-encoding method. In distribution GKM (BGKM), the secret key is shared between operator through equal- primitive based trait. We can use systematic codes in order to change the protected record. We can increase the number of operators participating in BGKM circle and can improve their access to supervised prescription. The operators will be capable of accessing information which are designated to its domain by using the code affiliated with it.

Apart from that, the study by Li et al. states that, using deduplication method, we can minimize the memory of the data, hence making the key administration more amenable. Convergent encryption makes use of a converged key to encipher the details which is achieved by means of the hash value of the file copy to the substantial stage. Here, operators safeguard the keys with themselves and then transfer the encrypted files to the online cloud storage. De-key improves our

file memory as well as prevents our records from being distorted and enables an additional backup storage.

Based on the study conducted by Binbusayyis and Zhang, using cipher text-policy-ABE system, an authorization access assistance can be provided to the users to minimize the load acting on the centralized storage cloud and authority. ABE secures the privacy of our data. Therefore, Tysowski suggested a framework where the keys will be segregated into certain portions and a part of it are saved in the cloud which will be automatically discarded based on the dispatched timer or user's action. Rather than sharing documents Immediately, we distribute the keys of the encoded message to them to minimize the risk and to protect from threat assaults by malicious users online.

Lastly, to achieve a concise understanding of the advantages of using decentralized cloud for key management, the following table can be helpful:

Responsibility for Storage of Keys	Advantages	Disadvantages
Centralized	Utilizes the scalable computational and network resources of the cloud. Relies upon the direct user-to-cloud link.	Requires trust in the cloud provider to not decode encrypted user data stored on its servers.
Centralized in a Trusted Authority that is Outside of Cloud Domain	Does not require trust in cloud provider. May control to cloud data as an intermediary node.	Requires maintenance of a scalable authority server by the client, or trust in a third-party guardian as a paid service.
Fully Decentralized Among Users	Requires no additional network elements. Key sharing may utilize cheap local links such as Wi-Fi or Peer-to-Peer Wired Network. Allows higher availability with greater bandwidth and cheaper.	Obtaining keys may require arbitration by an authority which entails additional traffic.

Table 1: Showing Merits and Demerits of using Different types of Cloud Infrastructure

2.3 Existing Steganography Methods

Nevertheless, countless studies and groundwork have been carried out for the literature in order to secure the data in the cloud. In this partition, majority of the latest steganographic methods which are enforced to safeguard the information in the cloud are demonstrated. More specifically, in (Sarkar & Chatterjee,2014) the authors schemed an attainable and productive Steganographic mechanism to further enhance the security on Data-at-rest. Every time a data or an information is stored in the data center of the cloud, not everyone is authorized to view the first content of it since an appropriate identification will be required. Through specific security and implementation investigation, the proposed prototype almost ensured the integrity of data when it was established in the data center of a Cloud Service Provider (CSP). However, this method is only capable of dealing with just a bounded measure of security threats in a small environment.

The researchers (Saini & Sharma, 2014), merged three algorithms and improved a stable security system of data protection in cloud computing, basically for certification and verification of the data (DSA), is exerted. Hence, a regular (AES) algorithm was implied and finally, the utmost phase of this mechanism was to cloak the data within an audio file by using Steganography method to provide the most ultimate security to the data. This mechanism satisfies both protection and verification but since it is a one by one method, the time entanglement is elevated.

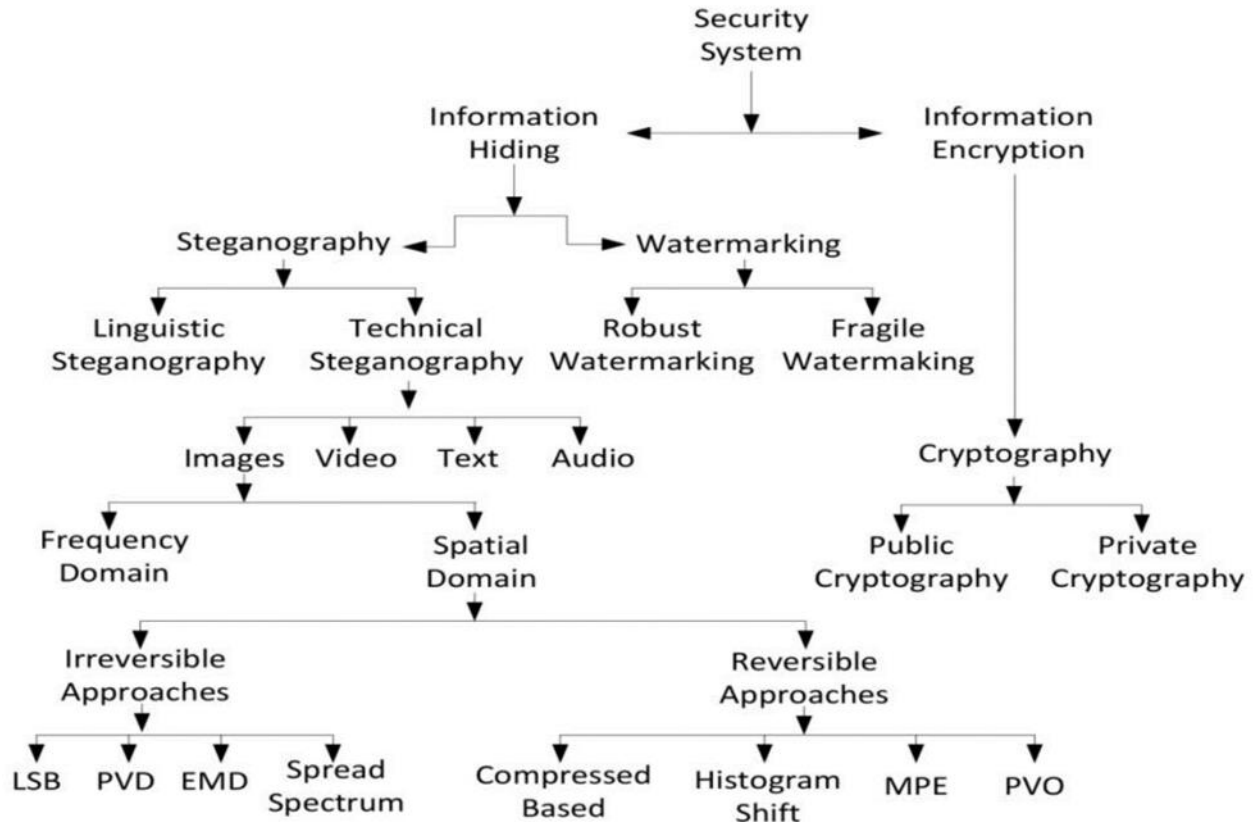


Figure 2: A Hierarchy of Different Types of Steganography

Here of, (Saravana Kumar & Arun, 2014) to provide security protection for cloud computing, the writers developed an algorithm in order to enhance a client owned security design. This algorithm has the aptitude for transmitting the encoded data to the distributor. By implying the similar algorithm method, the distributor will be able to apply security to the client's data via encryption. Therefore, the client's data remains protected at either end. The projected algorithm along with steganography uses ASCII and BCD security which saves the encoded piece of information in an image file. Likewise, they use the Common Deployment Model (CDM) algorithm beyond the cloud which provides practical security assistance. As of the proposed algorithm, the prime objective is in an encoded format to send and manipulate by the client to the distributors. The distributor as well keeps up the data information with a security algorithm from an unapproved admittance to ensure the privacy of the data is maintained.

In accordance to (Pant, Prakash, & Asthana, 2015), the writers contemplated to use steganography and cryptography scheme with each other for safeguarding of data. They assumed that RSA algorithm is the best secured algorithm compared to the others and in concern to improve the security to their data, the other algorithms merged with RSA algorithms. An image can be encrypted by going through the Steganography process, wherein the image will still appear to be identical for a human eye after the encryption process. Likewise, as a summary of the analysis presented by (Nimmy & Sethumadhavan, 2014), for cloud computing the writers projected an important authentication process to validate the verification by implying some additional security features such as password changing choices, session key mutuality among the operators and the cloud server, and mutual authentication. This suggested method of using steganography and personal distribution, presents a creative way to validate the authorization. Out-of-band verification enables the users to communicate easily which is also better in a way that we won't be needing a supplementary software or hardware for the end user.

Another excellent method by (Geetu Gopi and Rajkumar, 2017) has been suggested where a combination of compression and LSB method has been conducted. In the method Discrete Wavelet Transformation (DWT) technique has used to compress image after embedding data into the LSB of the cover image.

Algorithm used to insert data into the cover image:

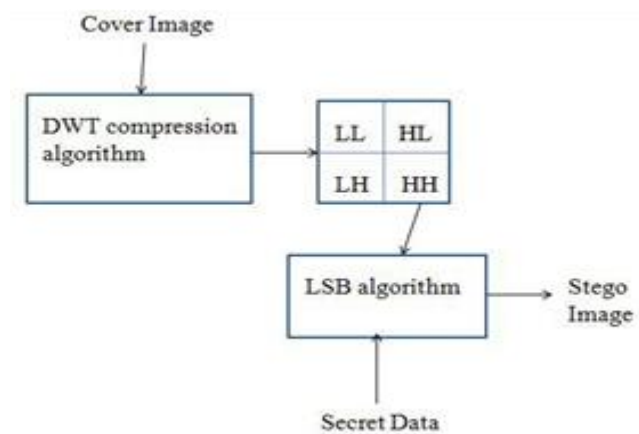


Figure 3: LSB Algorithm's Work Flow

Based on their research the embedding algorithm can be projected as below:

1. Read the cover image.
2. Read the secret information.
3. Based on the intensity value of the cover image, set n number of significant bits of the cover image to zero by using bit and operation.
4. Select first element (first character of a word or first pixel of an image) of the secret information.
5. The element is bit and with $a=128$. If the product is equal to a. Then add cover image pixel value after bit and operation with a value n using bit or operation.
6. If the product is not equal to a. Then divide a and a is bit and with the element.
7. Select next element of secret data.
8. Repeat step 5 until all elements of the secret data was embedded.
9. Display the stego image.

Algorithm used to extract secret data from stego image:

1. Read the stego image.
2. Based on the pixel value find the number of embedded bits.
3. Find the product of pixel value and number of embedded bits using bit and operation.
4. If the product is equal to number of embedded bits. Then find the secret data using bit or operation.
5. Display the secret data.

DWT Compression Algorithm:

Algorithm used to compress the cover image:

1. Read the cover image.
2. Separate image horizontally. Addition of two nearby pixels provides lower coefficients. Repeat addition for entire rows. Difference of two nearby pixels gives the higher coefficients.
3. After finding the 1D transform. Separate the 1 D transform of the image vertically. Find detail and approximation coefficients using addition and subtraction of the nearby pixels of 1D transform.
4. Display the 2D transform of the cover image.

Block diagram of 1D and 2D transform is shown as below:

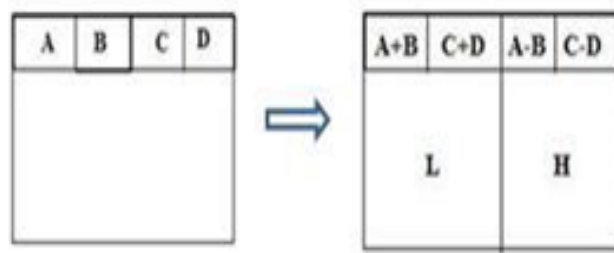


Figure 4:Block diagram of I D transform.

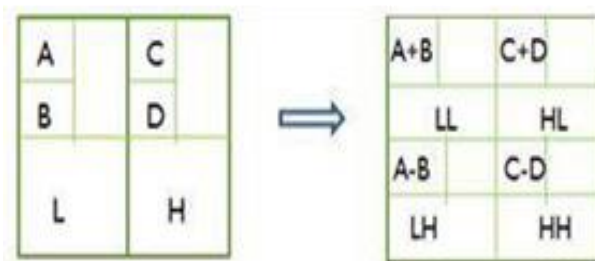


Figure 5:Block diagram of 2 D transform

2.4 Comparison of Existing and Proposed Steganographic Method

In addition, a comparison has been done between the existing and proposed LSB techniques as shown below:

Image Steganography Techniques	Description	Advantages
Extension of LSB (Least Significant bit)	Compression algorithm is used to maximize storage capacity.	Robust and efficient for hiding text and works efficiently for .bmp images.
LSB Compression	Comparison of Proposed & Existing Techniques	High image embedding capacity, sufficient payload and high security.
LSB and DCT (Discrete Cosine Transform)	Comparative Analysis of two techniques based on security, PSNR.	Peak signal to noise ratio is improved using LSB but security wise DCT is best.

Modified LSB	Hides secret message based on searching about identical bits.	More efficient, simple, Appropriate and accurate.
Combinations of LSB	Hiding the data in LSB bit pairs of pixels and comparison between two-bit pairs.	Less visible to human eye that is quality of image is better.
Out Proposed LSB Method	SHA256 processes the user given encryption key generate hush that is embedded into the LSB	Provides higher level of security by means of hashing keys that creates higher level of security after getting embedded into LSB of image.

Table 2: Comparison of Existing and Proposed LSB Algorithm

Chapter 3: Research Methodology

3.1 Introduction

The primary goal of the proposed system is to execute a secured key management system with non-vulnerable security mechanisms in a private peer-to-peer network as well as decentralized cloud. In order to provide strong security multiple encryption techniques have been integrated together with cryptography. For the encryption methods, Advanced Security Standard (AES), Secured Hash Algorithm 256 (SHA256) followed by Blocked Cipher Chain (CBC) has been used. On the other hand, Least Significant Bit (LSB) technique has been used for cryptography. All these schemes ensure data confidentiality, integrity, whereas, the decentralized cloud server will assure the availability of the data. The encryption key of the encoded file will be kept in the key management database to be easily backed up and recovered on demand. The strong library of Python programming language can easily accommodate all these objectives of the proposed system.

3.2 Key Management Service in Cloud

The cloud server will deal the management of the encryption keys in terms of creating, storing, saving and retrieving. The very basic working flow of the proposed system will be as below:

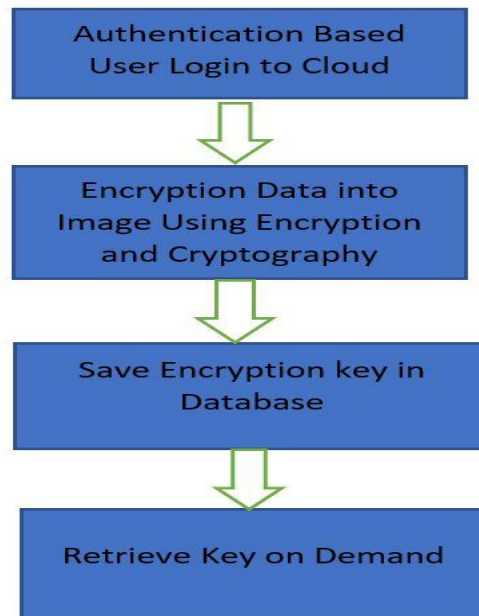


Figure 6: Basic Working Flow of Major Functions

At the beginning of the system startup, it will prompt user for inputting image, data file and password to encrypt it. Therefore, the AESCipher function from the Crypto library will convert the given plain password to cipher version. Each of the characters of these ciphered version of the password will be used by the SHA256 method during the encryption process of the data file containing secret information to be secured. Next, the random function will randomize the characters in the text file and pass it to LSB method that will embed all the information is the secret file into the given image.

A cloud server client setup will be done in order to implement the system. The proposed network will be a peer-to-peer network where the clients will be connected to the server by means wired technology. At first, the configuration the IP address for both server and clients will be implemented using IPv4. Socket features from python language will be used to create socket and establish connection between the server and clients. The overall network will look as shown below:

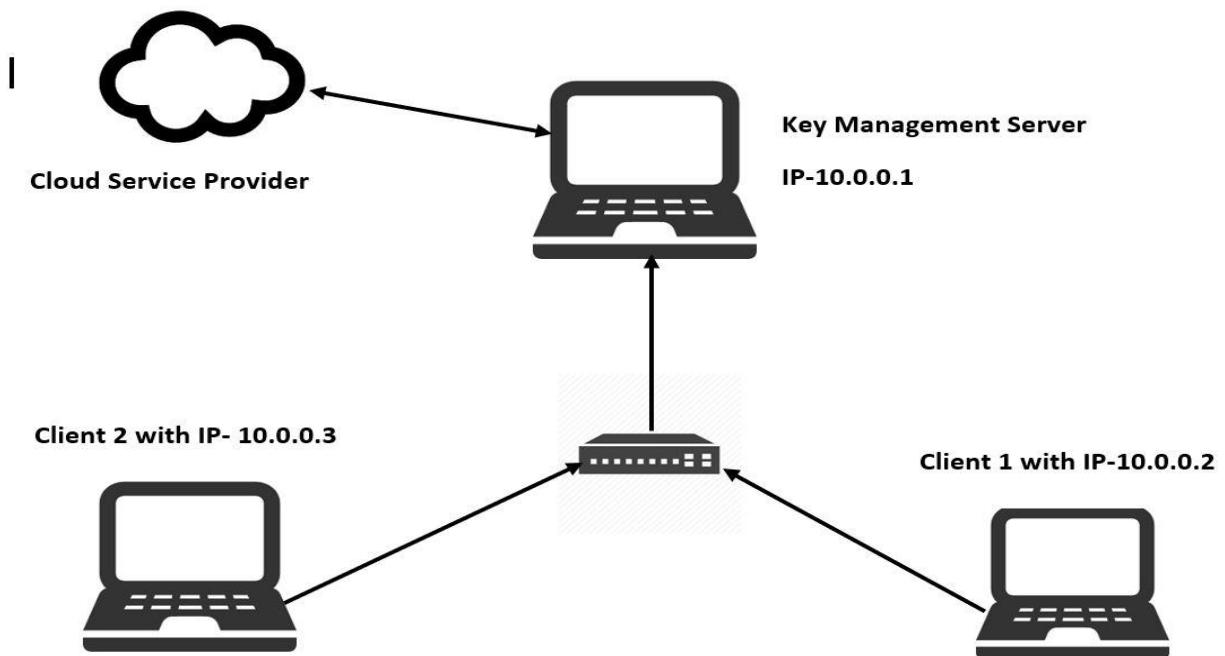


Figure 7: The Setup Structure of Server-Client in Proposed Cloud Network

3.3 Implementation and Testing

After successfully implementing the system as shown above, a series of testing will be conducted to assure a good quality system that will be able to run without errors. The system expected to run within a timely manner and less consummation of resources from the client's devices. The system should properly authenticate the clients during signing into the server, should provide a strong and secured combination of encryption and cryptographic techniques. The performance is supposed to be fast and the key generation and altering process should be strong. The server should be able to recover the encryption key if the client needs it. Hence, based on the must needed functionalities discussed above the required testing will be upon completion of the implementation process of the whole system.

3.4 Summary

The proposed system will be implemented in an organized manner by integrating strong methods that in terms will be tested as to assure quality as well.

Chapter 4: Project Methodology

4.1 Introduction

In this chapter, the research methodology of the proposed system as well as the minimum required configuration has been discussed step by step. Besides, a brief explanation on the source code of the system as well as how it works have also been provided. The system accomplishes its processes in several steps one after another. Firstly, a decentralized cloud server performs secure authentication of users to grant access to the KMS service. Furthermore, the system allows users to hide messages inside an image file that will be encrypted using a user given encryption key. These multiple levels of security mechanisms assure safe transmission of sensitive data. The basic methodology or work flow of the system can be illuminated by the following diagram -

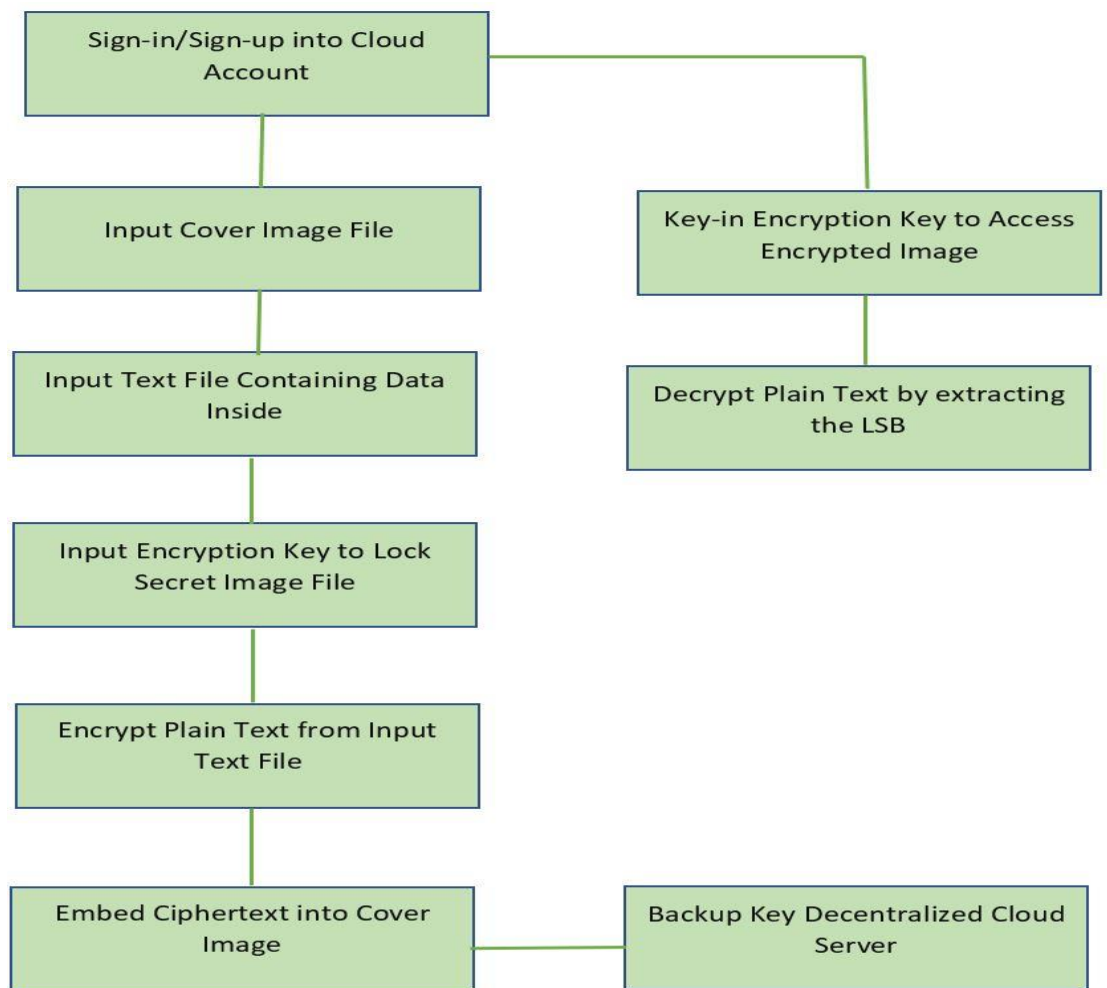


Figure 8: Overall Work Flow of Functions

4.2 Requirement and System Analysis

The proposed system has some specific requirements both in terms of software and hardware in order to operate efficiently. The necessary software and hardware requirements have been analyzed as demonstrated below:

4.2.1 Software Requirements

- Debian Based Linux Distributions (Recommended: Ubuntu 14.04 and 16.04)
- OpenStack (Open Source Cloud Structure Deployment)
- Python 2.6 and above

4.2.2 Hardware Requirements

Name of Components	No of Units Required
Computers (Min Required Configuration - Processor: Intel Core 2 Duo RAM: 4GB and above)	2
Ethernet Cable	1
Network Cards	2
Layer 2 Switch	1

Table 3: Hardware Requirements for Proposed Project

4.2.3 Functional Requirements

- Grant access to only authenticated users' during accessing cloud service.
- Capable of embedding plain text into the Least Significant Bits of Cover Image.
- Able to encode image file with strong encryption key
- Securely make transmission of encryption keys from client-pc to key management server.
- Accurately extract plain text from the encrypted and cloaked version image file.
- Able to view each stored encryption keys, search with file name and retrieve on demand.
- Properly display all the outputs with organized Command Line Interface (CLI).

4.2.4 Non-Functional Requirements:

- Proper direction to perform all the function with user manual.

4.2.5 System Flow Chart

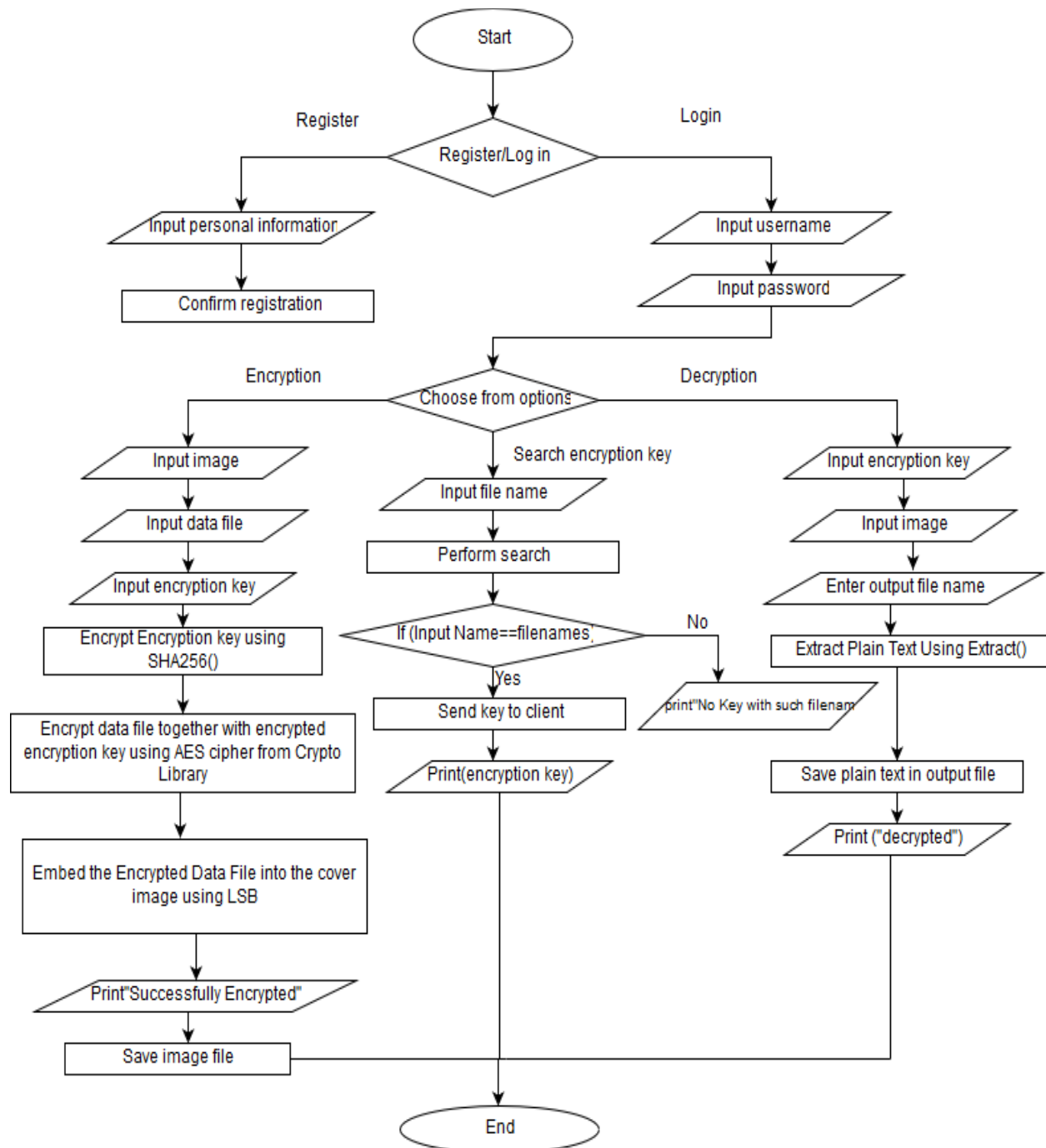


Figure 9: The Basic Functioning Flow of Proposed System

4.2.6 Description of System Flowchart

No	Name on Flowchart	Description
1	Start	Decentralized cloud-based key management system using steganography starts
2	Register/Login	Choose from the selection menu either from register or login
3	Input personal information	Enter information such as username, name, password, email
4	Confirm registration	A message showing the success of registration process
5	Input username	The user is required to enter registered username
6	Input password	User is required to enter correct password for a username
7	Encrypt	option encrypt from selection menu to start encryption process
8	Decrypt	Choose decrypt from selection menu to start decryption process
9	Search encryption key	Choose search encryption key from selection menu to start searching a key to decrypt that file
10	Input image file	Enter image file name as well its location in disk
11	Input data file	Enter data file name that contains secret information and needs to be encrypted
12	Input encryption key	Enter the encryption key that will be used to encrypt a specific image file
13	Encrypt encryption key using SHA256	The encryption key given by the user will be secured by using SHA256 and will be hashed
14	Encrypt data file together with encrypted encryption key using AES cipher of crypto library	System will automatically encrypt the file containing secret data and the hashed encryption key together using cipher that is a function of AES crypto library
15	Embed encrypted file into cover image using LSB	The data file with multiple encryption applied on it will be embedded into the cover image provided by user by using the least significant bits on it.
16	Save encrypted file	Store the encrypted image file in client's local disk.
18	If (input name == filename)	Checks whether the encryption key that the user is looking for exist in the database or not

19	Send key to client	If the key is found in the server database, it will be sent to the client.
20	Enter output file name	The client is needed to enter a file name where the extracted plain text will be written
21	Extract plain text using extract ()	The extract () function will be used to decrypt plain text from an image
22	Save plain text in output file	The written plain text in the output file will be stored on the client local disk drive.
23	Print “Encrypted”	Confirms user about the successful completion of encryption process
24	Print “Decrypted”	Confirms user that the decryption process has been complete successfully.
25	Print (encryption key)	The Encryption sent from cloud server will be printed in the client terminal to assure the client that the key has been found and successfully retrieved from the cloud database.
26	Print “No key with such name”	If the entered encryption key that the user is trying to find from server is not existing in the server then this message will be printed to inform.

Table 4: Description of the System Basic Functioning Flow Chart

The flowchart shown above projects the primary work flow and calling of several functions to perform different required tasks according to user’s demand. In addition, the table provided above illustrates the functions shown in flow chart in brief to provide the reader an easy understanding of the mentioned functions. However, the used functions belong to both user defined and library functions of python. The primary user defined functions that has been shown in the basic functioning flow chart are embed (), extract () and encrypt (). On the other hand, the primary library functions include SHA256 (), random () and AES () that have been imported from hash library, Crypto and Crypto.Cipher respectively. A couple of flow charts has been projected below in order to simplify the working process of user-defined functions embed () and extract () accordingly.

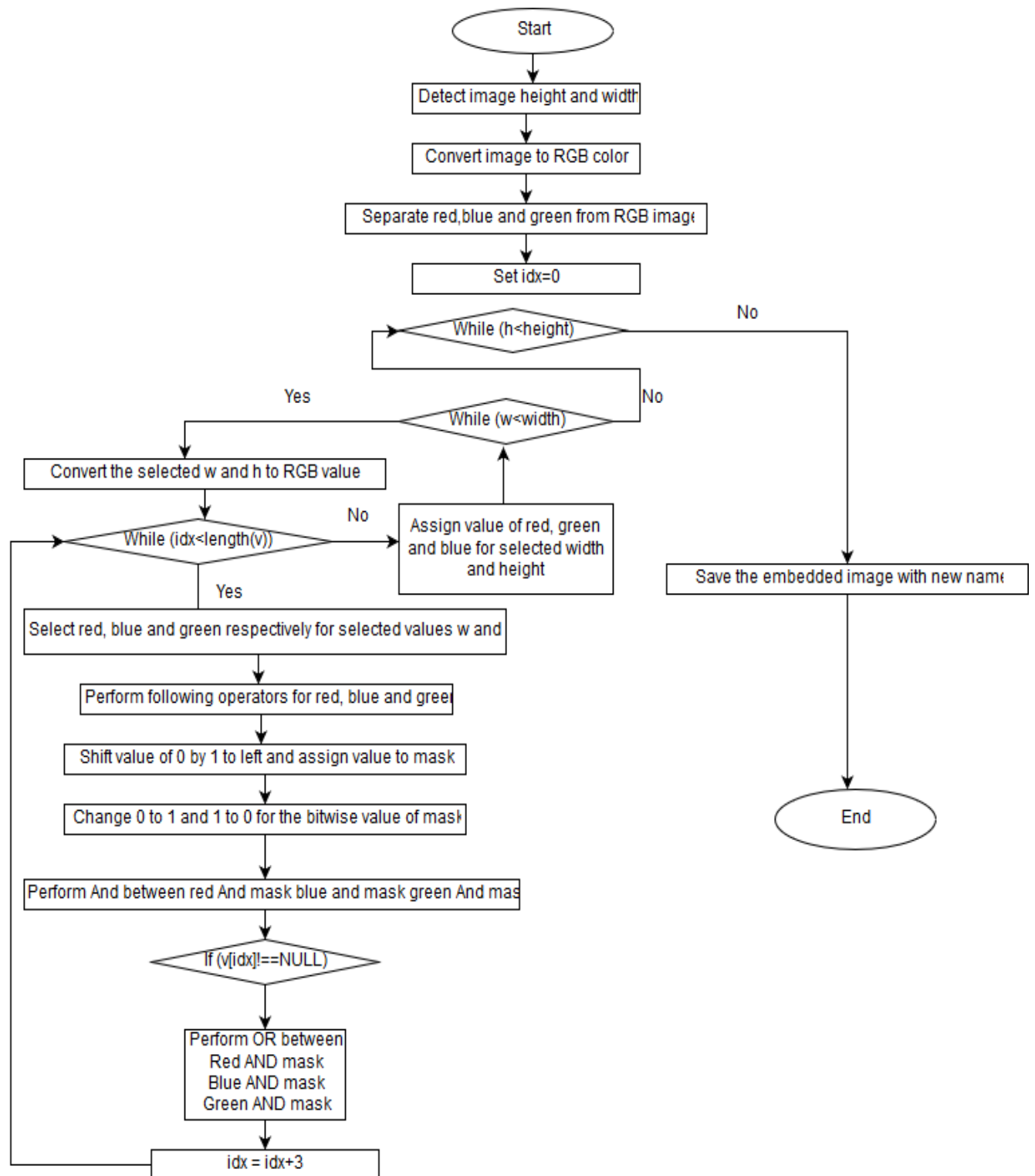


Figure 10: Flow Chart for Embedding Function

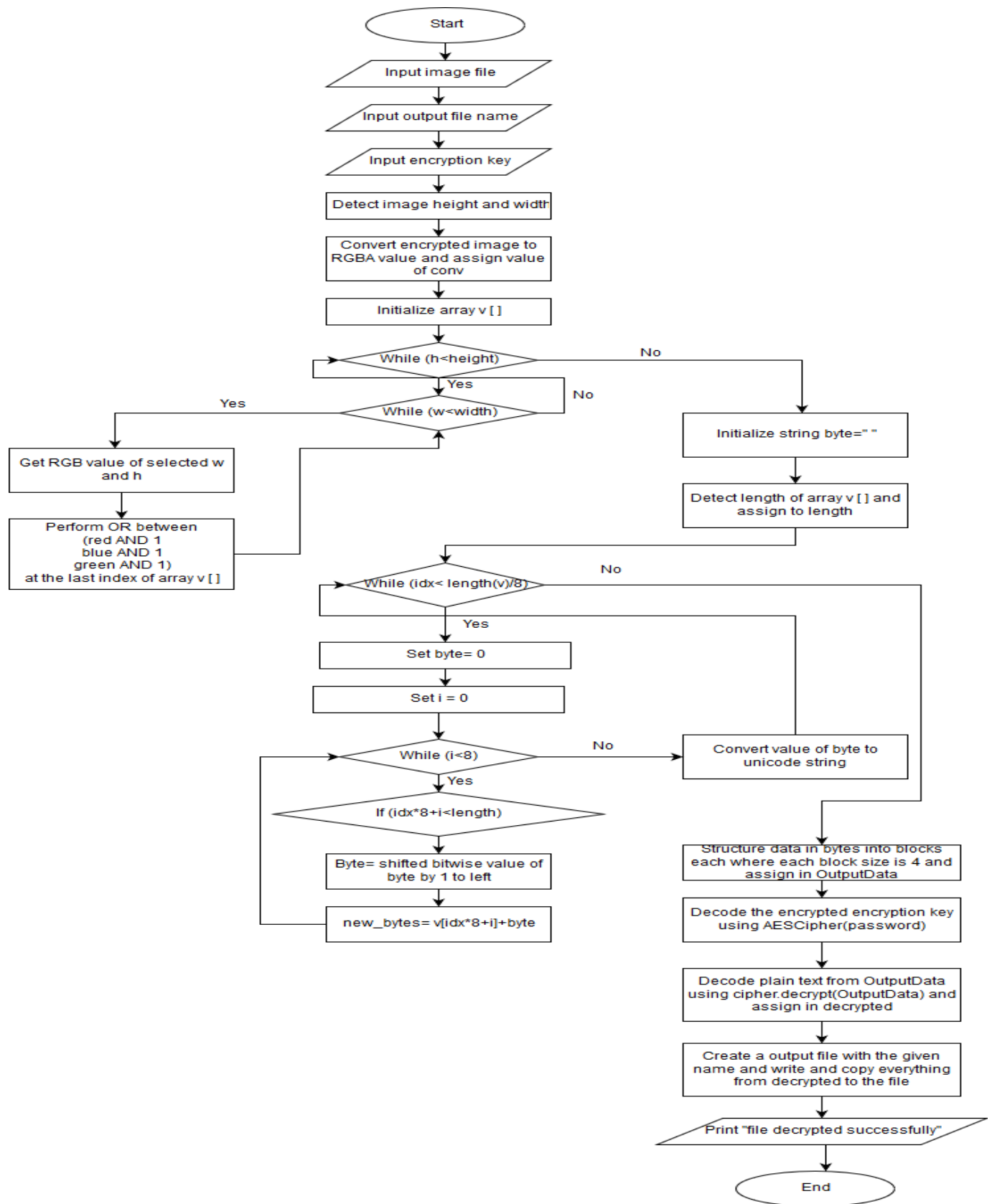


Figure 11: Flow Chart for Extract Function

4.2.6 Description of Project Flow Chart

At the beginning of the system, it is required to deploy a cloud network with at least two devices. One device will be used to run the server-side program and the other will be running client program intended to be used for the users. When the system starts running, the server will ask the client to input login information or to register if the user has not used the system earlier. Once successful login has been done, a selection menu will appear in the client interface asking to select one from menu. In a scenario of choosing the first option, the client will access encryption functions of the system. Therefore, user will require to input a cover image, file that is to be embedded into cover image and a password to encrypt the final encrypted file. The password will be hashed that will make it quite complex using SHA256 from hashlib. The program will then encrypt plain text using Cipher Block Chaining (CBC) technique where the hashed key will be used as the round key to add in each block. Therefore, LSB embedding technique will detect the least significant bits in all the pixels. By detecting their RGB value, an algorithm iv has been used to efficiently insert all the data into those LSB by setting them inside each color.

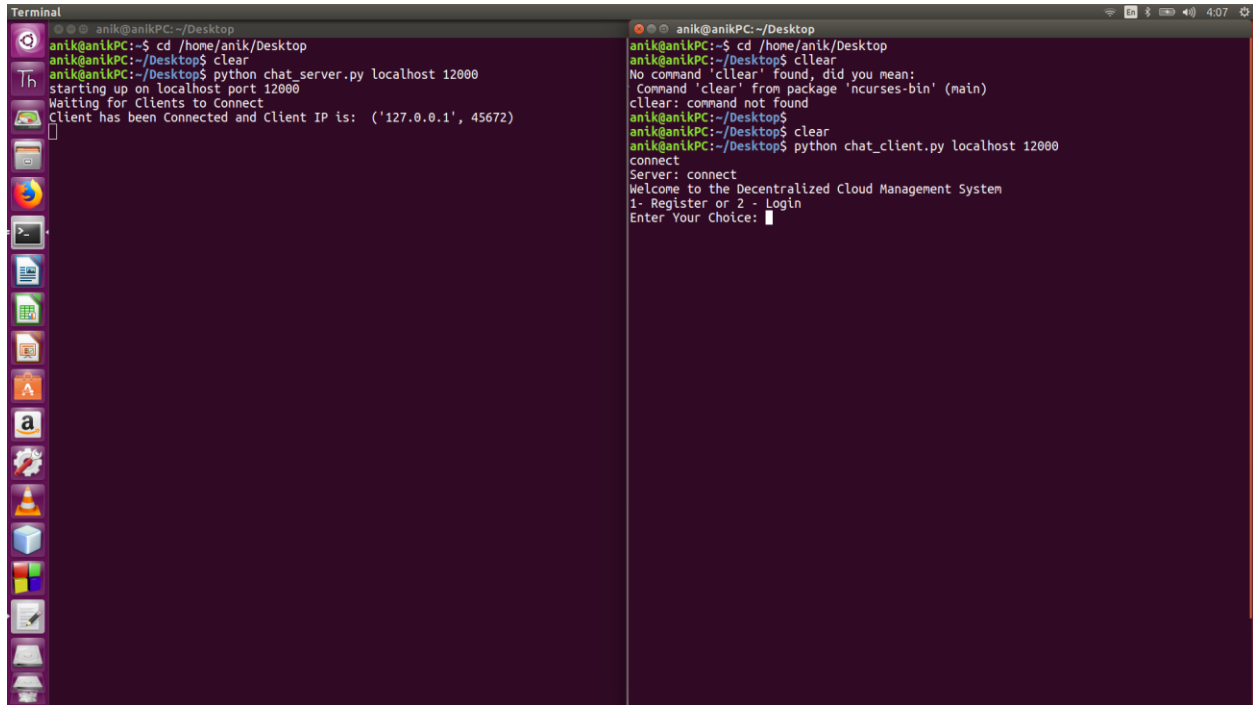
On the other hand, the decryption process on the selection menu will just take input of three of parameters- encrypted image, an output file name to write plain text and the encryption key to decode. Firstly, it will extract the text file from the image by reversing the encoding technique and therefore, the same library functions will perform to decrypt encryption key and plain text from the cipher version of data file. Finally, an output file will be created to save all the plain text.

The system automatically stores the encryption keys in the server to allow users to recover the keys on emergency situations. A file be created each time a new client signs up into the system to easily store all data and retrieve necessary portions of that data when required. The program will stop after completion of any single process such as performing encryption, decryption or searching encryption keys.

4.3 Design

In this section a detailed explanation regarding the system's design will be written. The system interface performing different functions of the system together with brief description about each scenario has been provided below:

4.3.1 System interface



The image shows two terminal windows side-by-side. The left window is the server terminal, and the right window is the client terminal. Both are running on a Linux desktop environment with a purple background and a sidebar of application icons.

```
anik@anikPC:~$ cd /home/anik/Desktop
anik@anikPC:~/Desktop$ clear
anik@anikPC:~/Desktop$ python chat_server.py localhost 12000
starting up on localhost port 12000
Waiting for Clients to Connect
Client has been Connected and Client IP is: ('127.0.0.1', 45672)
```

```
anik@anikPC:~$ cd /home/anik/Desktop
anik@anikPC:~/Desktop$ clear
No command 'clear' found, did you mean:
  Command 'clear' from package 'ncurses-bin' (main)
clear: command not found
anik@anikPC:~/Desktop$ clear
anik@anikPC:~/Desktop$ python chat_client.py localhost 12000
connect
Server: connect
Welcome to the Decentralized Cloud Management System
1- Register or 2 - Login
Enter Your Choice: 
```

Figure 12: Initial System Interface on Server and Client Terminal

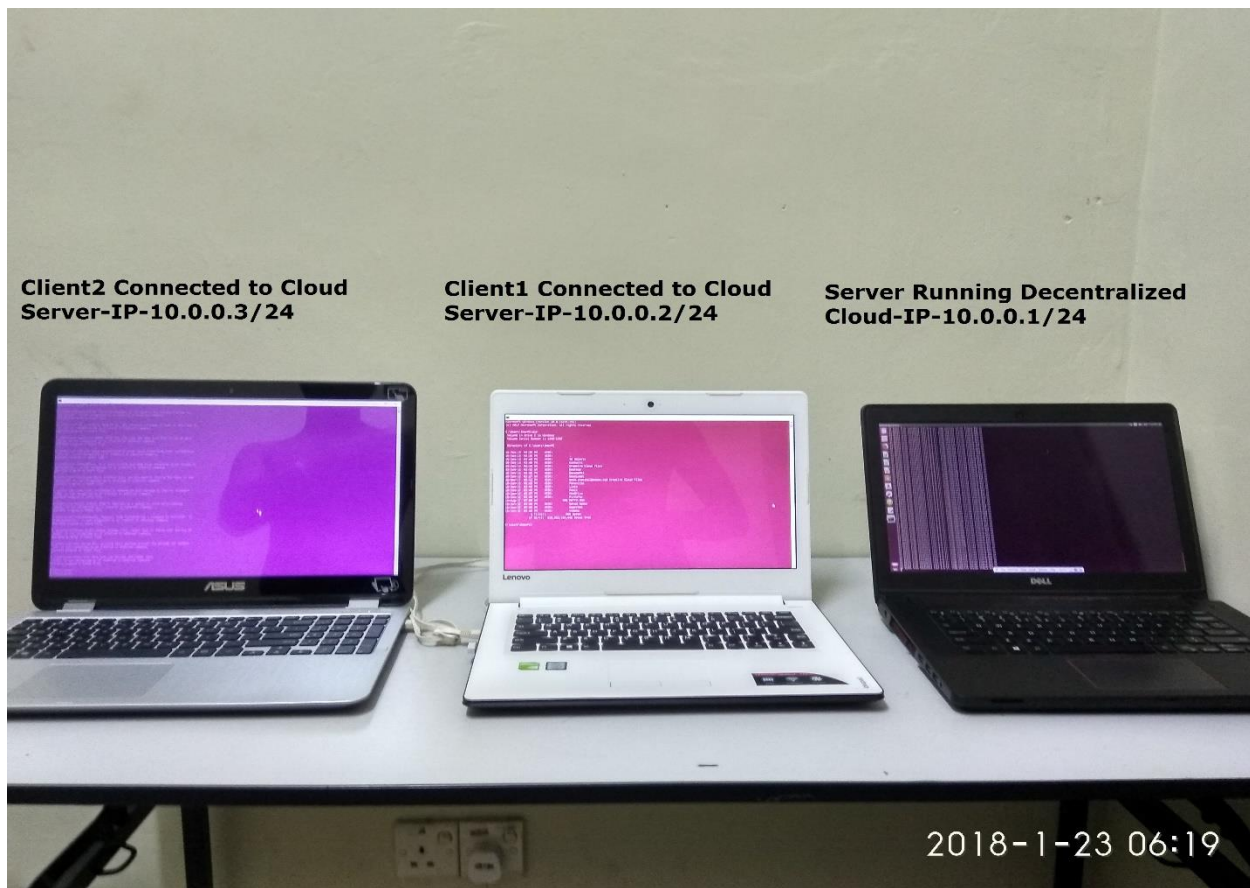


Figure 13: Decentralized Cloud Server-Client Setup

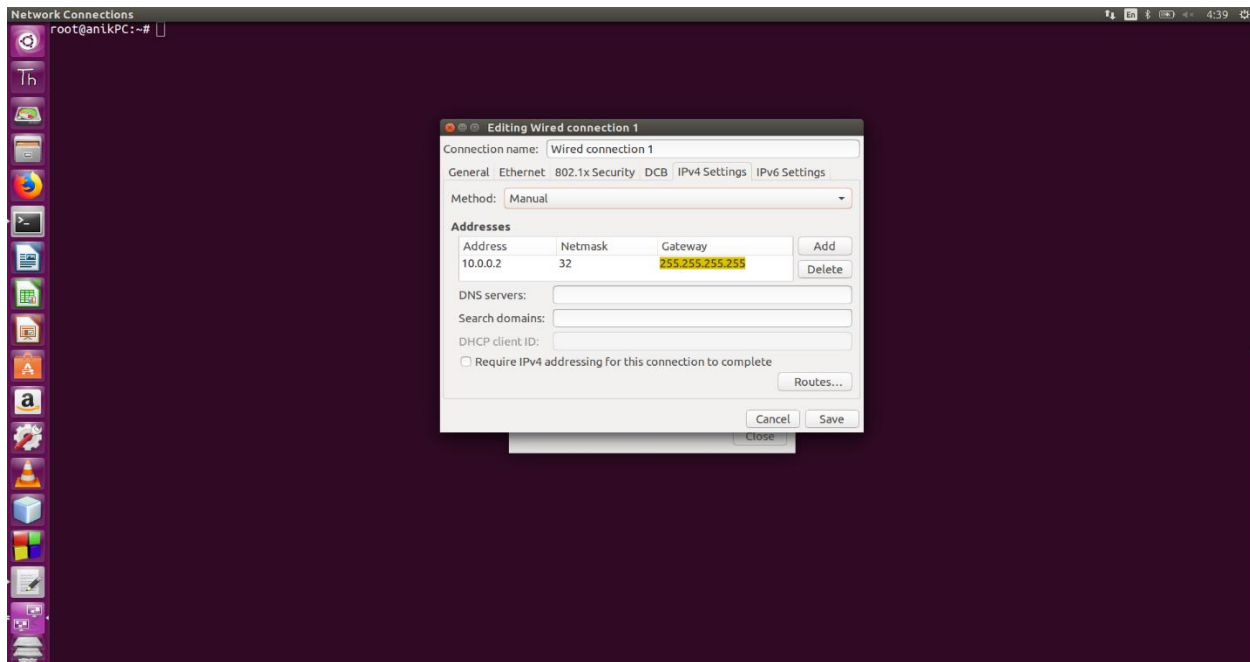


Figure 14: Setting Up IPv4 in Client Computer

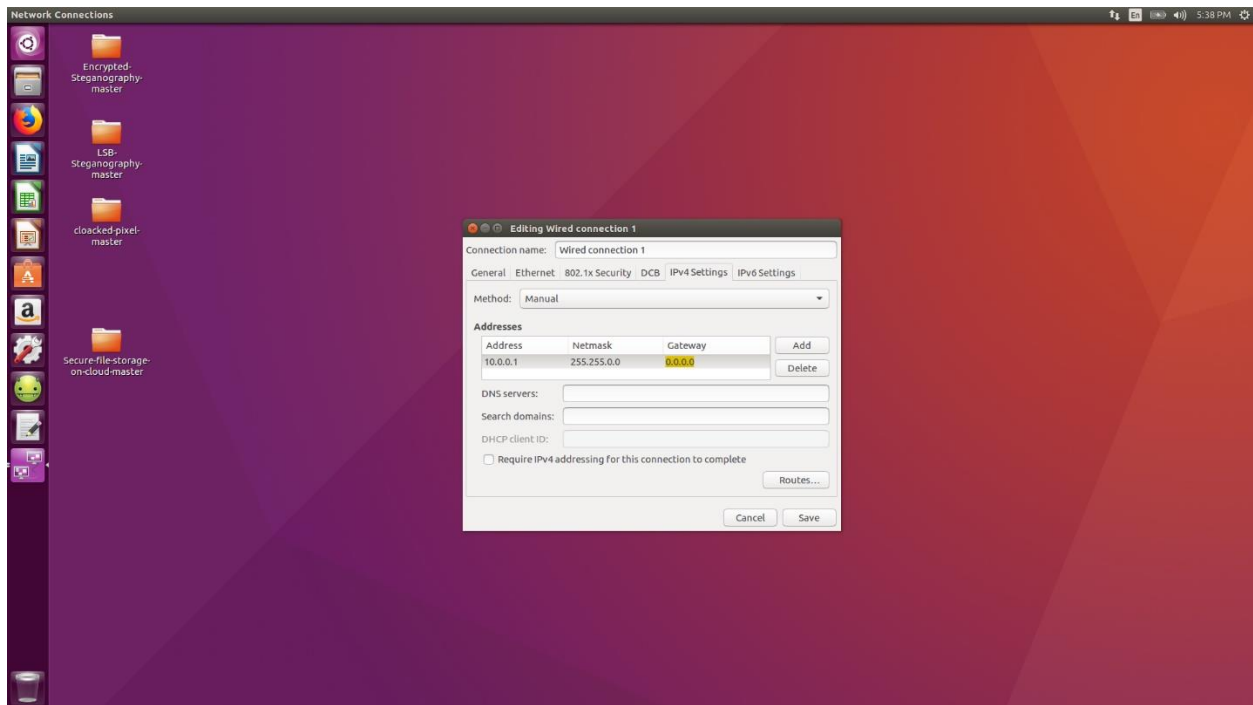


Figure15: Setting Up IPv4 in Server

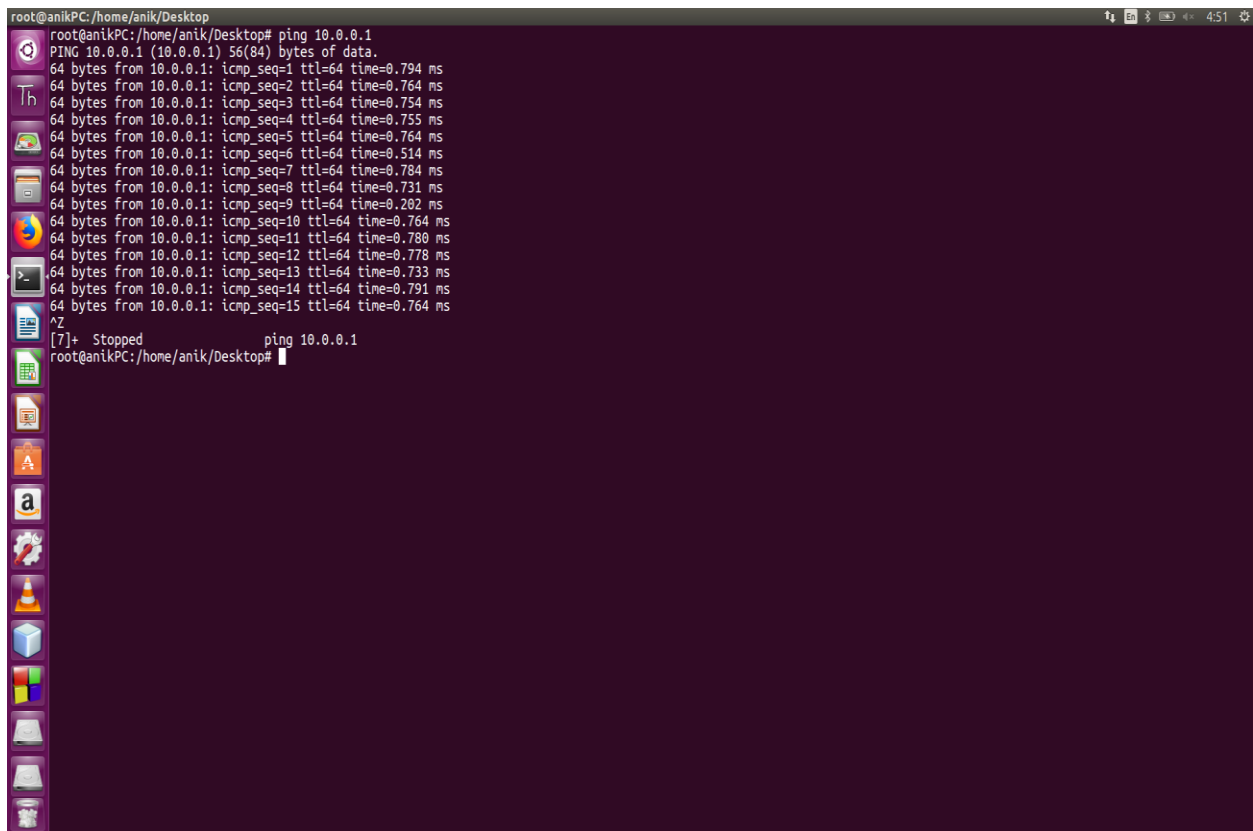


Figure 16: Showing that the Connection has been Established

```
root@anikPC:/home/anik/Desktop
root@anikPC:/home/anik/Desktop# python chat_client.py localhost 13000
connect
Server: connect
Welcome to the Decentralized Cloud Management System
1- Register or 2 - Login
Enter Your Choice: 1
Enter Your Name: ridoan
Enter Login Username: ridoan123
Enter Your Email Address: ridoankhan07@gmail.com
Enter Your Password: password123
Successfully Registered
Enter Your Choice: 2
Enter Username: ridoan123
Enter Password: password123
Successfully Logged-In
Choose a Your Service:
1 - Hide 2 - Extract 3 - Search Key
Enter Your Choice: 
```

Figure 17: Log-in Authentication from Server in Client Terminal

```
anik@anikPC:~/Desktop
anik@anikPC:~/Desktop$ cd /home/anik/Desktop
anik@anikPC:~/Desktop$ python chat_server.py localhost 25000
\starting up on localhost port 25000
Waiting for clients to connect
Client has been connected and client IP is: ('127.0.0.1', 59776)
```

Figure 18: Server Showing Confirmation of Client's Connection

```
anik@anikPC: ~/Desktop
anik@anikPC:~/Desktop$ python chat_client.py localhost 13000
connect
Server: connect
Welcome to the Decentralized Cloud Management System
1- Register or 2 - Login
Enter Your Choice: 2
Enter Username: ridoan123
Enter Password: password123
Successfully Logged-In
Choose a Your Service:
1 - Hide 2 - Extract 3 - Search Key
Enter Your Choice: 1
Enter Image File Name: orig.jpg
Enter the Data File containing Secret Message: secret.txt
Enter The Encryption Key to Encode the Data: hidemessage
[*] Input image size: 640x425 pixels.
[*] Usable data size: 99.61 KB.
[*] data size: 0.078 KB
[*] Encrypted data size: 0.113 KB
[*] %s embedded successfully!
The password has been saved successfully
Enter Your Choice: █
```

Figure 19: Encrypting and Embedding Message into Image and Saving Encryption Key

```
root@anikPC: /home/anik/Desktop
root@anikPC:/home/anik/Desktop# python chat_client.py localhost 15000
connect
Server: connect
Welcome to the Decentralized Cloud Management System
1- Register or 2 - Login
Enter Your Choice: 2
Enter Username: ridoan123
Enter Password: password123
Successfully Logged-In
Choose a Your Service:
1 - Hide 2 - Extract 3 - Search Key
Enter Your Choice: 2
Enter Encrypted Image File Name: orig.jpg-stego.png
Enter the Output File to Show Plain Text: output
Enter the Encryption Key: password
[*] Image size: 640x425 pixels.
[*] Written extracted data to output.
Enter Your Choice: █
```

Figure 20: Decrypting Plain Text from Embedded Image File

4.3.2 Implementation

The main implemented codes are provided below:

```
1 import socket
2 import sys
3
4 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5 server_name = sys.argv[1]
6 server_address = (server_name, 12000)
7 print >>sys.stderr, 'starting up on %s port %s' % server_address
8 sock.bind(server_address)
9 sock.listen(1)
10
11 while True:
12     print >>sys.stderr, 'Waiting for Clients to Connect'
13     connection, client_address = sock.accept()
14     try:
15         print >>sys.stderr, 'Client has been Connected and Client IP is: ', client_address
16         while True:
17             data = connection.recv(16)
18             data = 'Welcome to the Decentralized Key Management System'
19             #connection.send(data)
20             #connection.sendall('Welome to KMS')
21             #print >>sys.stderr, 'received "%s"' % data
22             if data:
23                 connection.sendall(data)
24             else:
25                 break
26         finally:
27             connection.close()
28
29
```

Figure 21: Socket Creation, Binding & Listening for Client to Connect

```
1 import socket
2 import select
3 import sys
4 import hashlib
5 import struct
6 import numpy
7 import matplotlib.pyplot as plt
8
9 from PIL import Image
10 from Crypto import Random
11 from Crypto.Cipher import AES
12
13 class AESCipher:
14
15     def __init__(self, key):
16         self.bs = 32 # Block size
17         self.key = hashlib.sha256(key.encode()).digest() # 32 bit digest
18
19     def encrypt(self, raw):
20         raw = self._pad(raw)
21         iv = Random.new().read(AES.block_size)
22         cipher = AES.new(self.key, AES.MODE_CBC, iv)
23         return iv + cipher.encrypt(raw)
24
25     def decrypt(self, enc):
26         iv = enc[:AES.block_size]
27         cipher = AES.new(self.key, AES.MODE_CBC, iv)
28         return self._unpad(cipher.decrypt(enc[AES.block_size:]))
29
30     def _pad(self, s):
31         return s + (self.bs - len(s) % self.bs) * chr(self.bs - len(s) % self.bs)
32
33     @staticmethod
34     def _unpad(s):
35         return s[:-ord(s[len(s)-1:])]
36
```

Figure 22: The Manipulation of the Library Functions Hashlib, Crypto.Cipher and Random


```

38 def decompose(data):
39     v = []
40
41     # Pack file len in 4 bytes
42     fSize = len(data)
43     bytes = [ord(b) for b in struct.pack("i", fSize)]
44
45     bytes += [ord(b) for b in data]
46
47     for b in bytes:
48         for i in range(7, -1, -1):
49             v.append((b >> i) & 0x1)
50
51     return v
52
53 # Assemble an array of bits into a binary file
54 def assemble(v):
55     bytes = ""
56
57     length = len(v)
58     for idx in range(0, len(v)/8):
59         byte = 0
60         for i in range(0, 8):
61             if (idx*8+i < length):
62                 byte = (byte<<1) + v[idx*8+i]
63             bytes = bytes + chr(byte)
64
65     data_size = struct.unpack("i", bytes[:4])[0]
66
67     return bytes[4: data_size + 4]
68

```

Figure 23: Decomposing and Assembling Function

```

# Set the i-th bit of v to x
def set_bit(n, i, x):
    mask = 1 << i
    n &= ~mask
    if x:
        n |= mask
    return n

```

Figure 24: Replacing RGB value of Image for a Selected Width and Height


```

77     # Embed data file into LSB bits of an image
78     def embed(imgFile, data, password):
79         # Process source image
80         img = Image.open(imgFile)
81         (width, height) = img.size
82         conv = img.convert("RGBA").getdata()
83         print "[*] Input image size: %dx%d pixels." % (width, height)
84         max_size = width*height*3.0/8/1024 # max data size
85         print "[*] Usable data size: %.2f KB." % (max_size)
86
87         f = open(data, "rb")
88         data = f.read()
89         f.close()
90         print "[+] data size: %.3f KB " % (len(data)/1024.0)
91
92         # Encrypt
93         cipher = AESCipher(password)
94         data_enc = cipher.encrypt(data)
95
96         # Process data from data file
97         v = decompose(data_enc)
98
99         # Add until multiple of 3
100        while(len(v)%3):
101            v.append(0)
102
103        data_size = len(v)/8/1024.0
104        print "[+] Encrypted data size: %.3f KB " % (data_size)
105        if (data_size > max_size - 4):
106            print "[-] Cannot embed. File too large"
107            sys.exit()
108
109        # Create output image
110        steg_img = Image.new('RGBA', (width, height))
111        data_img = steg_img.getdata()
112
113        idx = 0
114
115        for h in range(height):
116            for w in range(width):
117                (r, g, b, a) = conv.getpixel((w, h))
118                if idx < len(v):
119                    r = set_bit(r, 0, v[idx])
120                    g = set_bit(g, 0, v[idx+1])
121                    b = set_bit(b, 0, v[idx+2])
122                    data_img.putpixel((w,h), (r, g, b, a))
123                    idx = idx + 3
124
125        steg_img.save(imgFile + "-stego.png", "PNG")
126        savingstr=imgFile + "-stego.png", "PNG"
127
128        f= open("usr1.txt", "a+")
129        f.write(imgFile + "-stego.png", "PNG")
130        f.close()
131        print "[+] %s embedded successfully!"
132        print "The password has been saved successfully"
133

```

Figure 25: Applying AESCipher,Crypto and LSB Functions to Embed the Message into Image

```

def extract(in_file, out_file, password):
    # Process source image
    img = Image.open(in_file)
    (width, height) = img.size
    conv = img.convert("RGBA").getdata()
    print "[+] Image size: %dx%d pixels." % (width, height)

    # Extract LSBs
    v = []
    for h in range(height):
        for w in range(width):
            (r, g, b, a) = conv.getpixel((w, h))
            v.append(r & 1)
            v.append(g & 1)
            v.append(b & 1)

    data_out = assemble(v)

    # Decrypt
    cipher = AESCipher(password)
    data_dec = cipher.decrypt(data_out)

    # Write decrypted data
    out_f = open(out_file, "wb")
    out_f.write(data_dec)
    out_f.close()

    print "[+] Written extracted data to %s." % out_file

```

Figure 26: Extracting Image and Decrypting Plain Text Function

```

165 def find(str):
166     f=open("hello.txt")
167     text = f.read().strip().split()
168     while True:
169         if str == "":
170             continue
171         if str in text:
172             return 1
173             break
174         else:
175             return 2
176             continue
177     f.close()

```

Figure 27: Finding a Specific Word in File Database

4.5 Testing

Test Cases	Test Data	Expected Output	Actual Output
Test authentication process by trying to access with wrong password	Input wrong password frequently	The system prompts to re-enter password	The system prompts three times to enter password and rejects connection to client
Test LSB function with different sizes and quality of image	Input a series of image- 640 *480, 800*600, 1366*768.	The system should successfully perform fine with all image resolutions with .JPG and .PNG format	The system performs good with all of them except images with 1366 * 768 where system gets stuck sometimes
Test system by inputting different size of data file to check if it can process all	Input a number of text files containing different amount of data	It should be able to perform fine with image sizes below 2048 kilo bytes	The system nicely handles all data files sized below 2048 kilo bytes.
Test decrypt function with different encryption key	Input wrong encryption key for encrypted image	The file does not open unless correct encryption key is entered	The system keeps continuing until correct encryption is entered
Test key retrieve function with different keys	Input key number to search it from database	The system fetches and shows the encryption key	As long as the key was used before it outputs the key upon performing search
Test server by sending multiple request from client at the same time	Connect multiple clients to the server and perform operations simultaneously	The server might perform bit slow unless device has high configuration on it.	The system gets slow if more than five devices try to get service.

Table 5: Test System Functions and Check Results

Chapter 5: Results and Discussion

5.1 Introduction

In this section, the results of the implemented techniques in terms of performance, security and success rate have been discussed. These results can be used to prove the efficiency and scalability of the proposed system as well as the mechanisms used to implement it.

5.2 Results

A couple of cover images have been provided below to demonstrate that can help the users to have good faith on the system while using.



Figure 28: Cover Image Before Applying Code



Figure 29: Cover Image after Applying Methods with Hidden Data Embedded

It can be easily noticed that the difference between these images are not detectable by human eyes. However, the size data file containing secret message to be encrypted will be larger insignificantly due to having greater entropy. Therefore, the encrypted version of the data file before getting embedded will be containing random data.

Moreover, a professional software named Ben 4D has been used to analyze the processed cover photo with a view to detect existence of hidden data inside the image. The has been used to perform steganalysis as well as to find the vulnerability of the system. The portion of the logs scanned by the tool has been shown below:


```
Next position in scan buffer: Offset 0x001CC2D.1

*** Marker: EOI (End of Image) (xFFD9) ***
OFFSET: 0x0001CC2E

*** Searching Compression Signatures ***

Signature:      01557A9AE226A38386271DFE13D64298
Signature (Rotated): 0167FCEDBA3A8E8CF822163DB3564762
File Offset:    0 bytes
Chroma subsampling: 2x2
EXIF Make/Model: OK [NIKON] [NIKON D40]
EXIF Makernotes: NONE
EXIF Software:  NONE

Searching Compression Signatures: (3314 built-in, 0 user(*) )

EXIF.Make / Software      EXIF.Model      Quality      Subsamp Match?
-----
CAM:[Konica Minolta Camera, In] [DiIMAGE Z2] [ ] [ ] No
CAM:[Minolta Co., Ltd.] [DiIMAGE F100] [ ] [ ] No
CAM:[NIKON] [E2500] [ ] [ FINE ] No
CAM:[NIKON] [E4500] [ ] [ FINE ] No
CAM:[NIKON] [E5000] [ ] [ FINE ] No
CAM:[NIKON] [E8700] [ ] [ FINE ] No
CAM:[NIKON] [E885] [ ] [ FINE ] No
CAM:[OLYMPUS CORPORATION] [C8080WZ] [ ] [ ] No
CAM:[OLYMPUS OPTICAL CO.,LTD] [C2000Z] [ ] [ ] No
CAM:[OLYMPUS OPTICAL CO.,LTD] [C3040Z] [ ] [ ] No
CAM:[SEIKO EPSON CORP.] [PhotoPC 3000Z] [ ] [ ] No
CAM:[SONY] [DSC-H2] [ ] [ ] No
CAM:[SONY] [DSC-H7] [ ] [ ] No
CAM:[SONY] [DSC-H9] [ ] [ ] No
CAM:[SONY] [DSC-P200] [ ] [ ] No
CAM:[SONY] [DSC-S90] [ ] [ ] No
CAM:[SONY] [DSC-W7] [ ] [ ] No
SW :[IJG Library] [ ] [092] [ ]

The following IJG-based editors also match this signature:
SW :[GIMP] [ ] [092] [ ]
SW :[IrfanView] [ ] [092] [ ]
SW :[idImager] [ ] [092] [ ]
SW :[FastStone Image Viewer] [ ] [092] [ ]
SW :[NeatImage] [ ] [092] [ ]
SW :[Paint.NET] [ ] [092] [ ]
SW :[Photomatix] [ ] [092] [ ]
SW :[XnView] [ ] [092] [ ]

ASSESSMENT: Image is very likely processed/edited
```

Figure 30: Showing Analysis Report

5.3 Discussions

The key management system successfully generates encryption key with the given password. It also performs as long as the user would like to operate it. However, it has been noticed that the data containing file size increases slightly. Moreover, after copying the data into the image, the entropy changes a bit. It has been noticed that the frequency of zero and one in the least significant bits of the processed image is -0.5 on average basis, whereas, normal images have 0.5. Although the used methods will insert into the noisy portions of least significant bits, it is still possible to

detect their existence by searching least significant value with values having 0.5. The table below shows some of the properties found during the testing process.

Data File Size Before Encryption (in kilo bytes)	Data File Size After Encryption (in kilo bytes)	Entropy Change Rate
2.04	2.9	0.06
1.8	2.2	0.12
1.2	1.7	0.20
0.9	1.3	0.008
2.5	3.3	0.132

Table 6: Showing Image Stat Before and After Encryption

5.4 Summary

Most of the functions of the system are performing well. The data containing files sizes increase by 35% compared to the size of the original file. The change in entropy is high that provides benefit by making look almost like a series of random characers.

Chapter 6: Conclusion

6.1 Introduction

In this chapter, the limitations of the proposed system have been discussed based on the testing and the results found. A section has been provided to describe the future works that can be conducted.

6.2 Limitations of the System

The system is performing fine and the proposed objectives has also been completed. However, some functions are not working when the system runs for longer. The cloud server consumes higher resources and bandwidth when serving multiple clients concurrently. However, it was found that the key retrieve process does not work in times. Also, the key management has certain lacking that comprises periodic-key destruction, providing master key to access encryption keys, key regeneration and so on. Since, the data file gets encrypted before getting embedded into the image and it uses SHA256 scheme that increases the data file size slightly. In addition, an image steganography can only accommodate small data files and cannot be used for big sized data. Lastly, the authentication process in the system does not maintain high standard as it doesn't use any higher level secured protocols. However, the strong hybrid concealing techniques makes it almost impossible for the intruders to crack it.

6.3 Benefits

The key management system on cloud can be highly beneficial both for an individual and organizations where security is the main concern. Organizations that are required to share sensitive information of customer frequently can be greatly benefitted by using the proposed system. The decentralized cloud will allow to business organizations to setup peer-to-peer to network where the privacy of data sharing can be greatly performed by integrating the proposed system in automated systems usually in banks, forensics etc.

6.4 Future Works

The following criteria can be the point of research focus in the future-

- Providing steganographic features by means of sound and video files.

- Include more encryption techniques in the system and generate perform random combination of the methods that to make harder to be guessed by the intruders.
- Integrating more features in key management server to make it capable to serve the users on the move.

References

- Akhtar N, Khan S, Johri P. An improved inverted LSB image steganography. 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT). 2014. doi: 10.1109/ICICT.2014.6781374
- Mandal S, Bhattacharyya S. Secret data sharing in cloud environment using steganography and encryption using GA. 2015 International Conference on Green Computing and Internet of Things (ICGCIoT).2015. DOI: 10.1109/ICGCIoT.2015.7380699
- Lei S, Zishan D, Jindi G. Research on key management infrastructure in cloud computing environment. Grid and Cooperative Computing (GCC), 2010. 9th International Conference on. IEEE; 2010
- Damgård I, Jakobsen TP, Nielsen JB, Pagter JI. Secure Key Management in the Cloud. Oxford, UK: IMACC; 2013
- Wang Y, Li Z, Sun Y. Cloud Computing Key Management Mechanism For Cloud Storage; 2015.
- Shabir M, Iqbal A, Mahmood Z, Ghafoor A. Analysis of classical encryption techniques in cloud computing. Tsinghua Sci Technol 2016;21(1): 102-13
- Kumar NS, Lakshmi GV, Balamurugan B. Enhanced attribute based encryption for cloud computing. Procedia Compute Sci 2015;46:689-96
- Fakhar F, Shibli MA. Management of Symmetric Cryptographic Keys in Cloud Based Environment; 2013
- Binbusayyis A, Zhang N. Decentralized Attribute Based Encryption Scheme with Scalable Revocation for Sharing Data in Public Cloud Servers; 2015.
- Chen Y, Yang G. Efficient and Secure Group Key Management Based on EBS and Attribute Encryption; 2011
- Jia H, Chen Y, Mao X, Dou R. Efficient and Scalable Multicast Key Management Using Attribute Based Encryption; 2010
- Tysowski PK, Hasan MA. Cloud-Hosted Key Sharing Towards Secure and Scalable Mobile Applications in Clouds; 2013
- Li J, Chen X, Li M, Li J, Lee PP, Lou W. Secure deduplication with efficient and reliable convergent key management. IEEE Trans Parallel Distrib Syst 2014;25(6):1615-25
- Mandai, S., & Bhattacharyya, S. (2015). Secret Data Sharing in Cloud Environment Using Steganography and Encryption Using GA. 2015 International Conference on Green Computing and Internet of Things (ICGCIoT), 1469–1474.<https://doi.org/10.1109/ICGCIoT.2015.7380699>

- Nimmy, K., & Sethumadhavan, M. (2014). Novel Mutual Authentication Protocol for Cloud Computing using Secret Sharing and Steganography. 2014 Fifth International Conference on the Applications of Digital Information and Web Technologies (Icadiwt), 101–106
- NIST. “SHA-256 Example.”
<http://csrc.nist.gov/groups/ST/toolkit/documents/Examples/SHA256.pdf>.
- A.M.AL-Shatnawi, “A New method in Image Steganography with improved Image Quality”, Applied Mathematical Sciences, vol6, pp 3908-3915, 2012
- Mr. Ashwin Chandra C, Ms. Dharani S, “Decentralised Access Control with Aggregate Key Encryption For Data Stored In Cloud”, 2014
- Saravanakumar, and C. Arun. "AN EFFICIENT ASCII-BCD BASED STEGANOGRAPHY FOR CLOUD SECURITY USING COMMON DEPLOYMENT MODEL." Journal of Theoretical and Applied Information Technology 65.3 (2014)
- Amiya Nayak, Senior Member, IEEE, Milos Stojmenovic, Member, IEEE and Sushmita Ruj, Member, IEEE, “Decentralized Access Control with Anonymous Authentication of Data Stored in Clouds”, 2014
- International Journal of Engineering Trends and Technology (IJETT) – Volume 45 Number 6 - March 2017

Appendices

Secured Decentralized Cloud-based Key Management System Using Steganography

Ridoan Khan Anik 00013967

Department of Computing, Nilai University, Negeri Sembilan, Malaysia.



Abstract

Cloud Computing has brought revolutionary changes in the current context networking and organizations are getting more dependent on it continuously. Security is the biggest issue and massive research on it going on the different parts. In this paper, a key management system has been proposed and implemented that has been integrated with a series of robust security schemes on a decentralized cloud-based network. The combination of encryption and cryptographic techniques in the system makes it unique and non-vulnerable. The decentralized peer-to-peer cloud network has been deployed in SaaS infrastructure to allow clients to store data without the help of dedicated server. Least Significant Bit method along with SHA256 has been combined to encrypt sensitive data into image that will be encode with a password given by the user. The encryption keys responsible to give access to concealed data will be stored in server to enable instant retrieval of the key. Therefore, the effective use of these hybrid methods will allow the users to transmit data anywhere without having any concern of the data integrity and confidentiality.

Problem Statement

The rapid development as well as the increased number of people interested in cloud computing has created the requirement to solve the security vulnerabilities immediately. The key management system is a must to handle these increasing number of users since key management is most complex issue in the context of cryptography. Whereas, a proper management of these encryption keys is needed with easy to use interface with strong security protocols to prevent the attackers from getting un-authorized access to the other's data in a cloud-based server as well as to prevent the intruders to make brute-force attack. The rapid increasing size of organizations as well as the increased number of devices shows the importance of a secured key-management system implemented with hybrid techniques to stop intruders to guess the system's functioning process.

Objectives

The other primary objectives of the proposed system include-

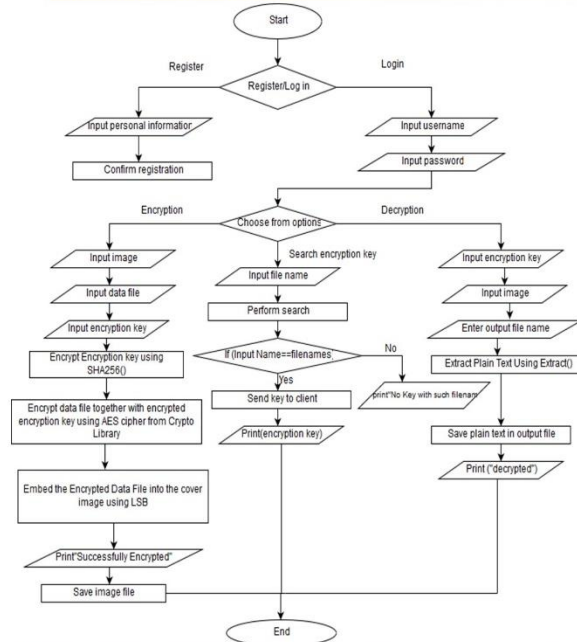
- To develop a key management server with secured login authentication on decentralized cloud server to allow effectively backup and recovery of encryption keys on demand.
- To implement a secured command line tool for client to use combined cryptographic techniques (LSB and SHA256) to encode/decode data in image with user given password to ensure data integrity and confidentiality during transmission.

Literature Review

Comparison Between Various Steganography Algorithms	Base	Steganography Without Cryptography	Steganography with Other Cryptography Techniques	Steganography with Our Cryptography Techniques
1	Security	One Level Security	Two Level Security	Two Level Security
2	Key Size	No Key Present	Fixed Size of Key	Random Size of Key
3	Steps Involve in Encryption of Data	No Step	Fixed Step	Depends on Key Size
4	Vulnerability Possibility	No Need	Possible	Very Hard to Crack

Responsibility for Storage of Keys	Advantages	Disadvantages
Centralized	Utilizes the scalable computational and network resources of the cloud. Relies upon the direct user-to-cloud link.	Requires trust in the cloud provider to not decode encrypted user data stored on its servers.
Centralized in a Trusted Authority that is Outside of Cloud Domain	Does not require trust in cloud provider. May control to cloud data as an intermediary node.	Requires maintenance of a scalable authority server by the client, or trust in a third-party guardian as a paid service.
Fully Decentralized Among Users	Requires no additional network elements. Key sharing may utilize cheap local links such as Wi-Fi or Peer-to-Peer Wired Network. Allows higher availability with greater bandwidth and cheaper.	Obtaining keys may require arbitration by an authority which entails additional traffic.

Research Methodology



Results and Discussions

Test Cases	Test Data	Expected Output	Actual Output
Test authentication process by trying to access with wrong password	Input wrong password frequently	The system prompts to re-enter password	The system prompts three times to enter password and rejects connection to client
Test LSB function with different sizes and quality of image	Input a series of image- 640 *480, 800*600, 1366*768.	The system should successfully perform fine with all image resolutions with JPG and PNG format	The system performs good with all of them except images with 1366 * 768 where system gets stuck sometimes
Test system by inputting different size of data file to check if it can process all	Input a number of text files containing different amount of data	It should be able to perform fine with image sizes below 2048 kilo bytes	The system nicely handles all data files sized below 2048 kilo bytes.
Test decrypt function with different encryption key	Input wrong encryption key for encrypted image	The file does not open unless correct encryption key is entered	The system keeps continuing until correct encryption is entered
Test key retrieve function with different keys	Input key number to search it from database	The system fetches and shows the encryption key	As long as the key was used before it outputs the key upon performing search
Test server by sending multiple request from client at the same time	Connect multiple clients to the server and perform operations simultaneously	The server might perform bit slow unless device has high configuration on it.	The system gets slow if more than five devices try to get service.

Conclusion/Summary

The decentralized cloud-based network allows dynamic database and also provides user-controlled scheme if the provider is untrusted. The use of the steganography together with an encryption method makes it quite complex for the hackers to break into a cloud server. Besides, that these systems have many limitations also when the intruder can guess the existence of data hidden inside the image. Besides, the system offers the user only to save the encrypted image and encryption in the local drive and only allows to upload the encryption key in the cloud server. In the future work, the can updated such that it can offer secure transmission of the file user to user within the system and also time validation for the encryption key as well. Lastly, it can be said that the key management system will have great potential.

Supervisor

Assoc. Prof. Dr. Abbas Mehdizadeh



Digital Receipt

This receipt acknowledges that **Turnitin** received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author: **Ridoan Khan Anik**
Assignment title: **Project Thesis**
Submission title: **Decentralized Cloud Based Key Ma...**
File name: **turnitin.docx**
File size: **2.75M**
Page count: **36**
Word count: **6,150**
Character count: **32,452**
Submission date: **23-Jan-2018 05:43PM (UTC+0800)**
Submission ID: **905736446**

Abstract

Cloud Computing has brought revolutionary changes in the current context networking and organizations are getting more dependent on it continuously. Security is the biggest issue and massive research on it going on the different parts. In this paper, a key management system has been proposed and implemented that has been integrated with a series of robust security schemes on a decentralized cloud-based network. The combination of encryption and cryptographic techniques in the system makes it unique and non-vulnerable. The decentralized peer-to-peer cloud network has been deployed in SaaS infrastructure to allow clients to store data without the help of dedicated server. Least Significant Bit method along with SHA256 has been combined to encrypt sensitive data into image that will be encode with a password given by the user. The encryption keys responsible to give access to concealed data will be stored in server to enable instant retrieval of the key. Therefore, the effective use of these hybrid methods will allow the users to use cloud service without having any fear of data integrity and confidentiality.

Background

A Key Management System (KMS) facilitates a dynamic way to generate, exchange, store and replace keys within a cryptosystem. It plays a potential role to provide security of the cryptographic keys typically between a system or user.

Encryption is combination of several mathematical rules that is usually used to restrict access of data as well as to make it accessible only by an authenticated entity. The sensitive data can be accessible by the intruders if not enough barrier is created. A key management system will fix the issues of security of encryption keys while in rest as well as on transit, secured distribution of keys within a system pair, proper backup and recovery of encryption keys and regenerate and destroy keys depending on the situation demands.

The key problem of a key management is to efficiently generate and secure encryption keys that is required to be executed each time a user attempts to communicate with another system usually by sending a message. A large number of user request to use encryption keys makes the job of