1.1INTRODUCTION:

Cloud computing is becoming more and more popular in the recent trend as it provides many benefits over the traditional data center solutions.

It is a style of computing in which IT related capabilities are provided “as a service”, allowing users to access a shared pool of configurable computing resources such as networks, servers, storage, applications that can be rapidly provisioned and released from the Internet (i.e., the Cloud) without knowledge of, expertise with, or control over the technology infrastructure that supports them.{Sarga, 2012 #1}

Even though cloud computing brings many benefits and reduces costs, there is still much distrust of such services from a security standpoint, and a lot of security concerns. Before moving to the cloud, organizations must have a well defined methodology for data migration. Cloud computing should be approached carefully and the data sensitivity should be taken in consideration. By adopting the cloud model, organization doesn’t have the control over their data, which a more classical model provides. By doing so, they trust the provider’s security policy. Security and privacy issues must be addressed at an early stage, because the subsequent changes could bring many complications, which can be expensive and risky.

1.2 RESEARCH PROBLEM:

The shared development environment presents some unique challenges, include those involving authentication, access control and authorization. Working with distributed applications and storage make data stored in cloud storage easily targeted by the masquerade attack and the insider data theft attack. These attacks threaten the data security and the data privacy of the stored data.

The major challenges that any cloud computing service provider must overcome are to make sure that if its servers are attacked by hackers, the client data cannot be stolen or misused and secondly the confidential client data must remain invisible even to the cloud service providers. The data needs to be encrypted and then sent to the

cloud provider. This means that the provider cannot operate on the data until it decrypts the same. The client has to provide private key to the server (has to compromise on confidentiality) for decryption and only then subsequent operations can be done on the data.

1.3PROJECTOBJECTIVES :

- The main objective of this project is to implement a cloud based application that use encryption techniques to provide data confidentially.
- Allow cloud provider to operate on the data in encrypted form without decrypt it.

1.4METHODOLOGY:

This thesis focuses on addressing security and privacy issues in the cloud by basically encrypting data before sending them to the cloud.

Our basic concept was to encrypt the data before send it to the Cloud provider, this means that the provider cannot operate on the data until it decrypts the same. This becomes a problem as the client has to compromise on confidentiality and share the private key to the server for decryption and only then subsequent operations can be done on the data. Also, this decryption process can be a performance issue for the service provider whenever it wants to operate on its client data.

This application will be a Cloud based Open Share point Website from where the users can store and manage their data onto the cloud. This data can be application data, personal data or computed data and will be in encrypted format.

1.5 ORGANIZATION OFRESEARCH:

Beside this chapter, this thesis consists of five chapters, as following:

Chapter 2: consists of two sections, first section discusses a background of cloud computing and the second one discusses previous studies that related to the thesis .

Chapter 3: discusses the methodology of the thesis .

Chapter 4: discusses tools and platforms and techniques witch used in the thesis .

Chapter 5: discusses implementation of the system and the results of this thesis .

Chapter 6: contains the conclusion and recommendation and the obstacles.

2.1 BACK GROUND

This chapter provides a brief overview about background of Cloud Computing. Section 2..11 provides the standard definition of Cloud Computing and its essential characteristics, services, deployment models respectively. The section 2.1.2 gives brief information about the Cloud Actors .Section 2.1.3 deals with the importance of security in the Cloud Computing, whereas section 2.1.4 provides the major security issues in Cloud.

2.1.1 What Is Cloud Computing?

Cloud is a computing model that refers to both the applications derived as services over the Internet, the hardware and system software in the datacenters that provide those services. Cloud Computing is treated as the high potential paradigm used for deployment of applications on Internet. This concept also explains the applications that are broaden to be accessible through the Internet. Cloud applications use large data centers and effective servers that host web applications and services.

2.1.1.1 Definition Of Cloud Computing

Cloud Computing is rapidly being accepted as a universal access appliance on the Internet. A lot of attention has been given to the Cloud Computing concept in deriving standard definitions. However, the definitions of Cloud Computing remain controversial. But here we have considered the standard definition which was given by the National Institute of Standards and Technology (NIST): *"Cloud Computing is model for enabling ubiquitous, convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction",{Maddineni, 2012 #2}* .

2.1.1.2 Essential Characteristics of Cloud Computing :

According to NIST, the Cloud model is composed of five essential characteristics:

- ***On-demand self-service****: A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider .*

- ***Broad network access****: Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations) .*
- ***Resource pooling****: The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. Examples of resources include storage, processing, memory, and network bandwidth .*

- ***Rapid elasticity*****:** Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

- ***Measured service:*** Cloud systems automatically control and optimize resource use by leveraging a metering capability (pay-per-use basis) at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service .

2.1.1.3 Service Models Of Cloud Computing

According to NIST, the cloud model is composed of three service models:

2.1.1.3.1Software as a Service (SaaS) :

In this model, software is delivered or offered to customers as a service. SaaS is a software application delivery model in which enterprises hosts and operates their application over the internet so that customers can access it. Earlier, companies have run software on their own internal infrastructures and computer networks, but now most of them have migrated to the SaaS model. One benefit of this model is customers do not need to buy any software licenses or any additional equipment for hosting the application. Instead, they pay for using the software application.

2.1.1.3.2 Platform as a Service (PaaS):

This model provides a platform for building and running custom applications. There is a lot of complexity and cost involved in building and running applications within the enterprise like support from hardware, a database, middleware, an operating system and other software. Enterprises should have team of network, database and system experts for setting up the configuration suitable for development. With continuous evolving business requirements, the application needs to be changed every time thus causing long development cycles due to redeployment. With PaaS, enterprises can build applications without installing any tools on their local systems and can deploy them without many difficulties. By using PaaS as a development platform web applications can be built almost five times faster than using conventional Java or .Net methods.

2.1.1.3.1Infrastructure as a Service (IaaS):

In organizations, maintaining their internal IT related tasks like installing, configuring servers, routers, firewalls and other devices is a cumbersome process and it requires dedicated personnel for carrying out these tasks. Apart from this there are many challenges the enterprise has to tackle while managing their infrastructure. IaaS provides a solution by migrating the IT infrastructure to the cloud and it is the responsibility of the cloud provider to tackle the issues of IT infrastructure management. Virtualization techniques are most commonly used in this model.

VMWare, Amazon EC2, IBM BlueHouse, Microsoft Azure, Sun ParaScale Cloud Storage, etc are some of the infrastructure services.{Annapureddy, 2010 #3}

2.1.1.4 Deployment Models of Cloud Computing :

According to NIST, the cloud model is composed of four deployment models:

- **Private cloud**: The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises .

- **Community cloud**: The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises .

- **Public cloud:** The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider .

- **Hybrid cloud:** The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds) {Maddineni, 2012 #2}.

2.1.2 Cloud Actors :

The NIST Cloud Computing Reference Architecture defines five different actors related to cloud computing: consumer, provider, auditor, broker, carrier.

1. **Cloud Provider :**

   “A person, organization, or entity responsible for making a service available to interested parties. A Cloud Provider acquires and manages the computing infrastructure required for providing the services, runs the cloud software that provides the services, and makes arrangement to deliver the cloud services to the Cloud Consumers through network access” .

2. **Cloud Consumer:**

   "A person or organization that maintains a business relationship with, and uses service from, Cloud Providers. A cloud consumer browses the service catalog from a cloud provider, requests the appropriate service, sets up service contracts with the cloud provider, and uses the service."

3. **Cloud Auditor:**

   “ A party that can conduct independent assessment of cloud services, information system operations, performance and security of the cloud implementation. A cloud auditor is a party that can perform an independent examination of cloud service controls with the intent to express an opinion thereon. A cloud auditor can evaluate the services provided by a cloud provider in terms of security controls, privacy impact, performance, etc. “

4. **Cloud Broker:**

"As cloud computing evolves, the integration of cloud services can be too complex for cloud consumers to manage. A cloud consumer may request cloud services from a cloud broker, instead of contacting a cloud provider directly. Hence the broker is an entity that manages the use, performance and delivery of cloud services, and negotiates

relationships between Cloud Providers and Cloud Consumers." Brokers provide three different types of services to the Cloud **Consumer**.

5. **Cloud Carrier :**

“ An intermediary that provides connectivity and transport of cloud services from Cloud Providers to Cloud Consumers. Cloud carriers provide access to consumers through network, telecommunication and other access devices.**”** {#4}

2.1.3 Importance of Security in Cloud Computing:

Security is an essential component of strong privacy safeguards in all online computing environments. Cloud customers and business providers both are willing to use online computing only if they trust that, their data will remain private and secure , because The information housed on the cloud is often seen as valuable to individuals with malicious intent. There is a lot of personal information and potentially secure data that people store on their computers, and this information is now being transferred to the cloud. {ASHALATHA, #5}

2.1.4 Important Security Issues in the Cloud :

Even though, the virtualization and Cloud Computing delivers wide range of dynamic resources, the security concern is generally perceived as the huge issue in the Cloud which makes the users to resist themselves in adopting the technology of Cloud Computing. Some of the security issues in the Cloud are discussed below:

Integrity: Integrity makes sure that data held in a system is a proper representation of the data intended and that it has not been modified by an authorized person. When any application is running on a server, backup routine is configured so that it is safe in the event of a data-loss incident. Normally, the data will backup to any portable media on a regular basis which will then be stored in an off-site location {#4}.

Availability: Availability ensures that data processing resources are not made unavailable by malicious action. It is the simple idea that when a user tries to access something, it is available to be accessed. This is vital for mission critical systems. Availability for these systems is critical that companies have business continuity plans (BCP‟s) in order for their systems to have redundancy {#4}.

Confidentiality: Confidentiality ensures that data is not disclosed to unauthorized persons. Confidentiality loss occurs when data can be viewed or read by any individuals who are unauthorized to access it. Loss of confidentiality can occur physically or electronically. Physical confidential loss takes place through social engineering. Electronic confidentiality loss takes place when the clients and servers aren't encrypting their communications {#4}.

2.2. INTRODUCTION :

This part demonstrates three previous research , all of this researches discuss security issues in cloud computing and some of them provide solutions .

2.2.1 SECURITY ISSUES AND USE OF CRYPTOGRAPHY IN CLOUD COMPUTING:

The paper {Sarddar, 2015 #8} discuss cloud security model for servicing to the consumers, security issues, analysis of vulnerabilities and attacks cloud computing frameworks, and the role of cryptography in cloud computing. to get insights a new security approach with the implementation of cryptography to secure a data at cloud data centers. So, the cloud data centers are assessed by the authenticated clients only and the approach takes a less time for execution and better security parameter.

2.2.1.1 Cloud Security Model For Servicing To The Consumers :

During communication process consumers are front end and cloud service providers are back end. For resource pooling various steps are included:

- User authentication and login process: In this web browser collects all necessary information about the consumer using various security technologies/protocols like SSL/SSH/TLS.
- Web browser provides all information to policy manager which authenticate the consumer using public key infrastructure, certification authority and others.

2.2.1.2 Use Of Cryptography In Cloud Computing :

In cloud computing the users can upload their information to the centralized large data centers where management of data and services are not trust worthy because information is uploaded by the users into cloud data centers not encrypted hence that is accessed by everyone and lead to above mentioned security challenges. For better security of cloud data centers the information is encrypted by the users by using cryptography techniques before uploading into the cloud data centers.

2.2.2 DATA SECURITY IN CLOUD COMPUTING WITH ELLIPTIC CURVE CRYPTOGRAPHY:

The paper {Gampala, 2012 #7} explore data security of cloud in cloud computing by implementing digital signature and encryption with elliptic curve cryptography.

The author consider the security of the traditional environment To preserve security of your cloud-based virtual infrastructure, you must perform security best practice at both the traditional IT and virtual cloud. To ensure data confidentiality, authentication, integrity, and availability, the provider should include the following solutions:

- Encryption: the sensitivity of data may require that the network traffic to and from the virtual machine be encrypted, using encryption at the host OS software.
- Physical security: keep the virtual system and cloud management hosts safe and secure behind carded doors, and environmentally safe.
- Authentication and access control: the authentication capabilities within your virtual system should copy the way your other physical systems authenticate. One time password and biometrics should all be implemented in the same manner. Also authentication requires while you are sending data or message from one cloud to other cloud. To provide message authentication we will use digital signatures.
- Separation of duties: as system get more complex, misconfiguration take place, because lack of expertise coupled with insufficient communication. Be sure to enforce least privileges with access controls and accountability.
- Configuration, change control, and patch management: this is very important and sometimes overlooked in smaller organizations. Configuration, change control, patch management, and updated processes need to be maintained in the virtual world as well as physical world.

- Intrusion detection and prevention: what's coming into and going out of your network has to know. A host based intrusion prevention system coupled with a hypervisor based solution could examine for virtual network traffic.

The proposed security solutions, is authentication and encryption for secure data transmission from one cloud to other cloud that requires secure and authenticated data with elliptic curve cryptography.

Proposed Procedure To Enhance Data Security In Cloud :

A and B act as public clouds with data, software and applications. A want to send data to B's cloud securely and data should be authenticated. We are here trying to send a secure data from A to B by applying digital signature and encryption to data with elliptic curve cryptography. if B wants document from A's cloud then B's user will place a request to A's user. A's user select corresponding document from A's cloud data storage and then apply the hash function, it will give message digest. Sign the message digest with his private key by using A's software. It is called digital signature. Encrypt digitally signed signature with B's public key using ECC algorithm. Encrypted cipher message will be send to B. B's software decrypt the cipher message to document with his private key and verify the signature with A's public key. using ECC algorithm. Encrypted cipher message will be send to B. B's software decrypt the cipher message to document with his private key and verify the signature with A's public key.

2.2.3 PRIVACY AND CONFIDENTIALITY ISSUES IN CLOUD COMPUTING ARCHITECTURES:

The master thesis{Jiménez Martínez, 2013 #6} discuss the privacy and confidentiality requirements, issues and challenges applied to the Cloud Computing paradigm, identify the existing evidence on this topic and establish relationships among works to find gaps and conflicting areas.

The methodology used was a systematic literature review, to provide the reader with a portion of the current research background in this field.

What is the Systematic Literature Review :

A systematic literature review (often referred to as a systematic review) is a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest. Individual studies contributing to a systematic review are called primary studies; a systematic review is a form of secondary study.

The review questions of the thesis:

- What is the impact of privacy and confidentiality requirements in Cloud Computing architectures?
- Which are the currently identified issues and challenges regarding privacy and confidentiality in Cloud Computing platforms. What are some of the solutions proposed to solve these issues?

3.1METHODOLOGY:

This thesis focuses on addressing security and privacy issues in the cloud by basically encrypting data before sending them to the cloud.

Then the data remain encrypted in the cloud and only users authorized by the data owner can get the credential for accessing the encrypted data. The encrypted data can be decrypted only after they are downloaded to an authorized user machine. In such a scenario, the privacy of the data does not depend on an implicit assumption of trust of the server or of the service level of agreement (SLA). Instead, the protection of privacy depends on the encryption techniques used to protect the data .the proposed schema was shown in Figure 3.1.

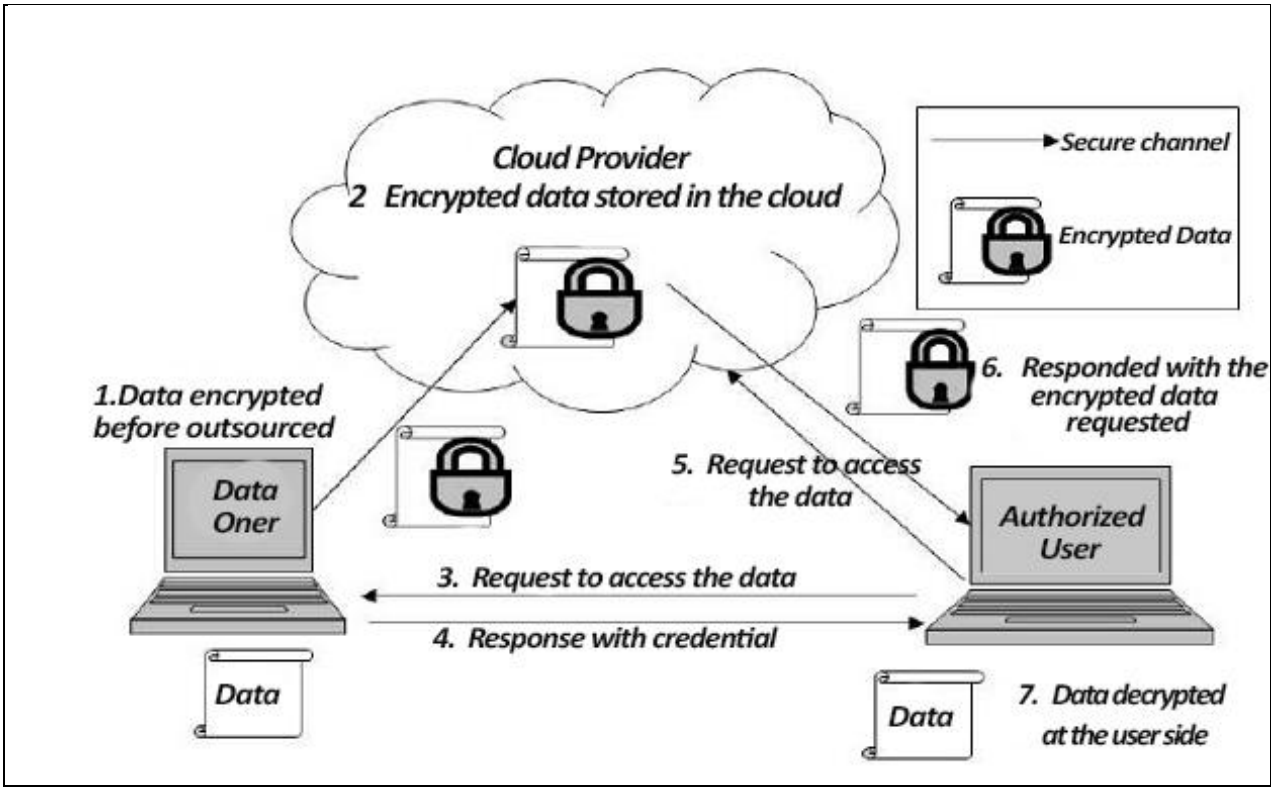

Figure 3.1:Basic Architecture for Preserving Data Privacy in the Cloud {Mohammad, 2014 #9}

When the data transferred to the Cloud we use standard encryption methods to secure the operations and the storage of the data. Our basic concept was to encrypt the data before send it to the Cloud provider.

But the last one needs to decrypt data at every operation. The client will need to provide the private key to the server (Cloud provider) to decrypt data before execute the calculations required ,which might affect the confidentiality and privacy of data stored in the Cloud.

3.2PROPOSED SYSTEM DESIGN:

The proposed system is cloud based platform which will be used by the user to upload data onto the cloud. An open share point website will be deployed onto the cloud for this purpose using java Enterprise Edition (JEE) technology. Once logged in successfully, the user will be prompted to encrypt the data before moving it onto the cloud and a private key will be entered by the user . This key will be used when the user decrypts final output file/data.

The proposed system provides the cloud data security using the homomorphic encryption technique. This technique provides functionality to perform operations on encrypted data without using the private key and without decrypting that data. After decrypting the result of this operation, it is the same as if we carried out the calculation on the plain data. This major strength of homomorphic encryption allows the cloud service providers to perform operations on encrypted client data without compromising on the client data privacy.

- Deploy private cloud computing environment using OpenStack.
- Design a web based application that can be deployed on the cloud and can provide functionality to secure user data.
- Develop the web application on MVC architecture using Java 2 enterprise edition.

- Homomorphic Encryption and decryption techniques are implemented using the Paillier's algorithm using the security packages provided by Java. These packages are also used for creation of the secrets keys.

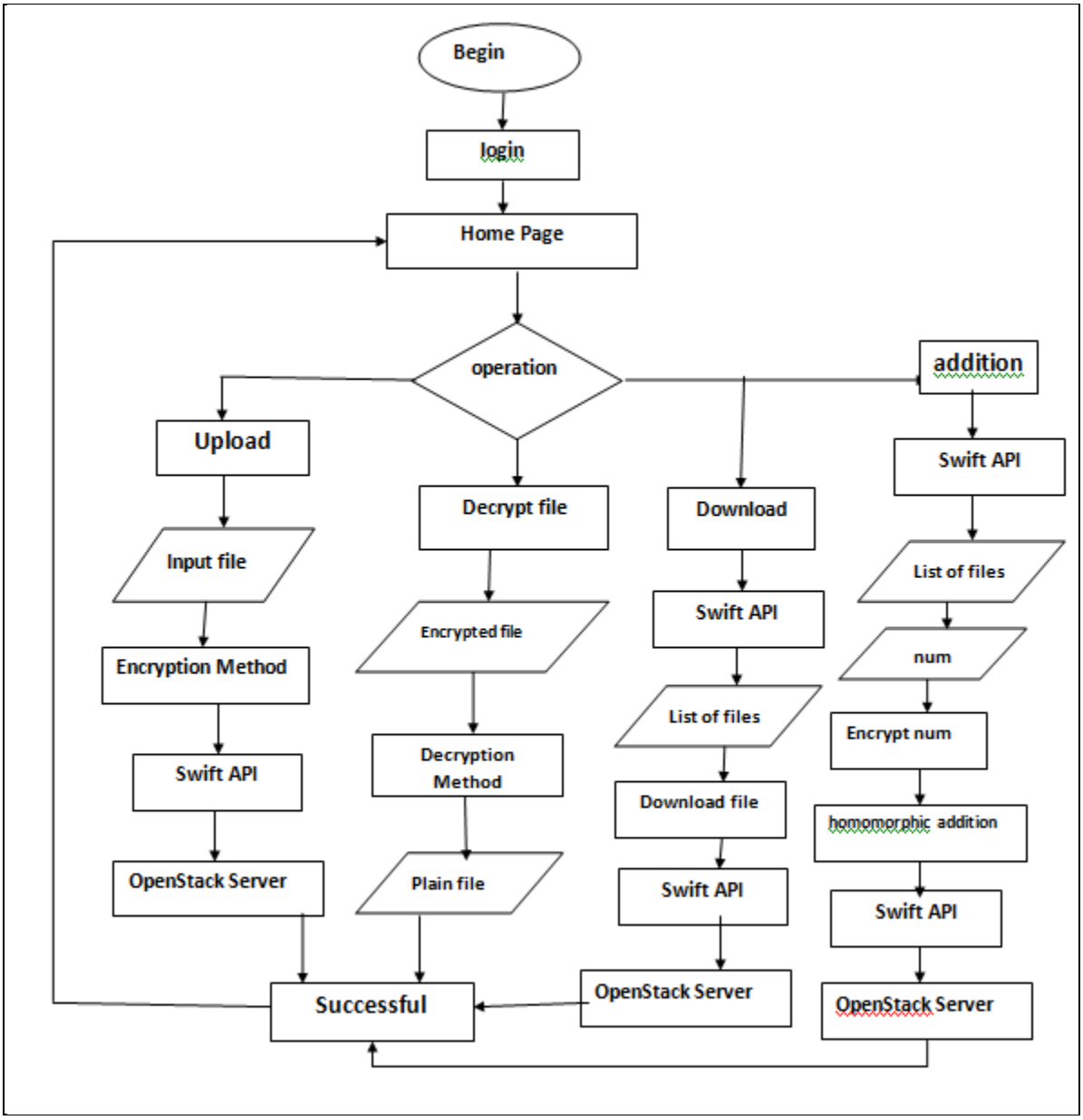

Figure 3.1: Proposed System Flow Chart Diagram

4INTRODUCTION:

This section discuss all tools and techniques used in this thesis.

4.1 TOOLS AND ENVIROMENTS:

4.1.1 Ubuntu :

Ubuntu is an open source Debian-based Linux distribution. Sponsored by Canonical Ltd., Ubuntu is considered a good distribution for beginners. The operating system was intended primarily for personal computers (PCs) but it can also be used on servers[10].

4.1.2 OpenStack:

The OpenStack project is an open source cloud computing platform that supports all types of cloud environments. The project aims for simple implementation, massive scalability, and a rich set of features. Cloud computing experts from around the world contribute to the project.

OpenStack provides an Infrastructure-as-a-Service (*IaaS*) solution through a variety of complementally services. Each service offers an application programming interface (*API*) that facilitates this integration.

4.1.2.1OpenStack services :

Table 4.1 list of 1OpenStack services

Service	**Project name**	**Description**
Dashboard	Horizon	Provides a web-based self-service portal to interact with underlying OpenStack services, such as launching an instance, assigning IP addresses and configuring access controls.

Compute	Nova	Manages the lifecycle of compute instances in an OpenStack environment. Responsibilities include spawning, scheduling and decommissioning of virtual machines on demand.
Networking	Neutron	Enables network connectivity as a service for other OpenStack services, such as OpenStack Compute. Provides an API for users to define networks and the attachments into them. Has a pluggable architecture that supports many popular networking vendors and technologies.
Storage		
Object Storage	Swift	Stores and retrieves arbitrary unstructured data objects via a *RESTful*, HTTP based API. It is highly fault tolerant with its data replication and scale out architecture. Its implementation is not like a file server with mountable directories.
Block Storage	Cinder	Provides persistent block storage to running instances. Its pluggable driver architecture facilitates the creation and management of block storage devices.
Shared services		
Identity service	Keystone	Provides an authentication and authorization service for other OpenStack services. Provides a catalog of endpoints for all OpenStack services.
Image Service	Glance	Stores and retrieves virtual machine disk images. OpenStack Compute makes use of this during instance provisioning.
Telemetry	Ceilometer	Monitors and meters the OpenStack cloud for billing, benchmarking, scalability, and statistical purposes.
Higher-level services		
Orchestration	Heat	Orchestrates multiple composite cloud applications by using

		either the native HOT template format or the AWS Cloud Formation template format, through both an OpenStack-native REST API and a Cloud Formation-compatible Query API.
Database Service	Trove	Provides scalable and reliable Cloud Database-as-a-Service functionality for both relational and non-relational database engines.

we will talk about the storage services because this research focus on it , to provide secure Storage to the users[11].

4.1.2.2Storage in OpenStack:

Storage is found in many parts of the OpenStack stack, and the differing types , all of this types It falls under two basic category ephemeral storage and persistent storage. This section focuses on persistent storage options .

Ephemeral Storage

If you deploy only the OpenStack Compute Service (nova), your users do not have access to any form of persistent storage by default. The disks associated with VMs are "ephemeral," meaning that (from the user's point of view) they effectively disappear when a virtual machine is terminated.

Persistent Storage

Persistent storage means that the storage resource outlives any other resource and is always available, regardless of the state of a running instance.

Today, OpenStack clouds explicitly support three types of persistent storage: *object storage*, *block storage*, and *file system storage*.

Object Storage

With object storage, users access binary objects through a REST API. Object storage is implemented in OpenStack by the OpenStack Object Storage (swift) project. OpenStack Object Storage provides a highly scalable, highly available storage solution by relaxing some of the constraints of traditional file systems.

Block Storage

Block storage (sometimes referred to as volume storage) provides users with access to block-storage devices. Users interact with block storage by attaching volumes to their running VM instances.

These volumes are persistent: they can be detached from one instance and re-attached to another, and the data remains intact. Block storage is implemented in OpenStack by the OpenStack Block Storage (cinder) project, which supports multiple back ends in the form of drivers. Your choice of a storage back end must be supported by a Block Storage driver.

Shared File Systems Service

The Shared File Systems service provides a set of services for management of Shared File Systems in a multi-tenant cloud environment. Users interact with Shared File Systems service by mounting remote File Systems on their instances with the following usage of those systems for file storing and exchange. Shared File Systems service provides you with shares. A share is a remote, mountable file system. You can mount a share to and access a share from several hosts by several users at a time. With shares, user can also:

- Create a share specifying its size, shared file system protocol, visibility level

- Create a share on either a share server or standalone, depending on the selected back-end mode, with or without using a share network.
- Specify access rules and security services for existing shares.
- Combine several shares in groups to keep data consistency inside the groups for the following safe group operations.
- Create a snapshot of a selected share or a share group for storing the existing shares consistently or creating new shares from that snapshot in a consistent way
- Create a share from a snapshot.
- Set rate limits and quotas for specific shares and snapshots
- View usage of share resources
- Remove shares.

Table 4.1 Comparison of differing types of storage in openStack [12]

	Ephemeral storage	Block storage	Object storage	Shared File System storage
Used to	Run operating system and scratch space	Add additional persistent storage to a virtual machine (VM)	Store data, including VM images	Add additional persistent storage to a virtual machine
Accessed through	A file system	A block device that can be partitioned, formatted, and mounted (such as, /dev/vdc)	The REST API	A Shared File Systems service share (either manila managed or an external one registered in manila) that can be partitioned,

				formatted and mounted (such as /dev/vdc)
Accessible from	Within a VM	Within a VM	Anywhere	Within a VM
Managed by	OpenStack Compute (nova)	OpenStack Block Storage (cinder)	OpenStack Object Storage (swift)	OpenStack Shared File System Storage (manila)
Persists until	VM is terminated	Deleted by use	Deleted by use	Deleted by use
Sizing determined by	Administrator configuration of size settings, known as *flavors*	User specification in initial request	Amount of available physical storage	• User specification in initial request • Requests for extension • Available user-level quotes • Limitations applied by Administrator
Encryption set by	Parameter in nova.conf	Admin establishing encrypted	Not yet available	Shared File Systems service does not apply any additional

		volume type, then user selecting encrypted volume		encryption above what the share's back-end storage provides
Example of typical usage	10 GB first disk, 30 GB second disk	1 TB disk	10s of TBs of dataset storage	Depends completely on the size of back-end storage specified when a share was being created. In case of thin provisioning it can be partial space reservation (for more details see Capabilities and Extra-Specs specification)

Object Store

The Object Store allows you to push a file directly to the cloud where it is stored thrice. You can then configure the file to be directly accessible on the web.

4.1.2.3 Encryption and Key Management

OpenStack Object Storage does not encrypt files before storing in a cluster . This means that if a user needs to store sensitive information in OpenStack, he will have to encrypt files before sending them to OpenStack and take care of key management for encryption himself.

Also users in role of Reseller Admin can view any file on any of the accounts, which is why if customers want to prevent provider's personnel from accessing their data, they should encrypt their files before uploading to OpenStack.

4.1.3J2EE (Java 2 Enterprise Edition)

J2EE stands for Java 2 Platform Enterprise Edition, it is a platform-independent, Java-centric environment from Sun. It is used for developing, building and deploying of online Web-based enterprise applications. The J2EE platform comprises of a set of services, APIs, and protocols that can be used for developing web based applications.

4.1.3.1Java Servlets:

Java Servlets are java programs written at server side. Web container of application server provides runtime environment for deploying java servlets. When the J2EE application server gets a client request, servlets are executed. Some of the features of Java Servlets are:

1) Security: Java Servlets inherits the security feature that the Web container provides.

2) Session Management: End user identity and state is kept intact across more than one requests.

3) Instance persistence: Frequent disk access is prevented. This enhances server performance.

4.1.3.2 JSP Technology

Work profiles of a Web designer and a Web developer are brought together with the help of JSP technology. HTML can be used by the web designer to design and the layout for a Web page. Then the Web developer can use Java code and other JSP related tags and work independently to write the business logic. Files are tied up by servlets to

handle the static presentation logic and the dynamic business logic independently. Also,

Java can be embedded directly into an HTML page with the help of JSP by using special tags. Extensive coding is involved in Servlet programming. Static code content and dynamic code content has to be identified and separated if any changes need to be made to the code to ease incorporation of the changes. This also allows both Web developers and the Web designer to work independently.[13]

4.1.4 Eclipse IDE

Eclipse is an integrated development environment (IDE). It comprises of a workspace and an extensible plug-in system. This IDE can be used to develop applications coded in Java. When certain plug-ins are added to the Eclipse IDE, we can then use it for making applications in C, C++, COBOL, JavaScript, Perl, PHP, Python, and Ruby and many other programming languages. The Java development tools are present in Eclipse software development kit (SDK) for developers.[14]

A built-in incremental Java compiler is provided in this IDE and a full model of the Java source files. This allows for advanced refactoring techniques and code analysis.

4.2 TECHNIQUES

4.2.1homomorphic encryption:

4.2.1definition:

Homomorphic encryption is the conversion of data into cipher text that can be analyzed and worked with as if it were still in its original form.

4.2.2 Introduction to Homomorphic Encryption

Homomorphic encryptions allow complex mathematical operations to be performed on encrypted data without compromising the encryption. In mathematics, homomorphic describes the transformation of one data set into another while preserving relationships between elements in both sets. The term is derived from the Greek words for "same

structure." Because the data in a homomorphic encryption scheme retains the same structure, identical mathematical operations -- whether they are performed on encrypted or decrypted data -- will yield equivalent results.

The concept of homomorphic encryption was first discussed by Rivest, Adleman And Dertouzos . They used some restricted classes of functions such as addition or multiplication in their discussion. Fully Homomorphic Encryption (FHE) methods that were recently developed have the capability to support all kinds of functions.

4.2.3 Homomorphic Encryption types:

Homomorphic Encryption systems can be classified into different types based on the operations that it allows on its raw data. Following sections discuss these different types of homomorphic encryption systems.

4.2.3.1 Partially Homomorphic Encryption Systems

Partially homomorphic encryption systems can support only a single operation on encrypted data as they are defined over a group. Different types of partially homomorphic Encryption systems are as follows.

4.2.3.2 Additive Homomorphic Encryption Systems

This type of system allows homomorphic computation of only the addition operation. The product of two cipher texts will decrypt to the sum of their plain texts.

$Enc(x + y) = Enc(x).Enc(y)$

generalization of Paillier system . These subsystems satisfy the property that the product of two cipher texts will decrypt to the sum of their plain texts.

$Enc(x + y) = Enc(x).Enc(y)$

4.2.3.3 Multiplicative Homomorphic Encryption Systems

This type of system allows homomorphic computation of only the Multiplication operation. Some well-known examples of multiplicative homomorphic encryption

systems are RSA algorithm and ElGamal encryption system . These subsystems satisfy the property that the product of the cipher texts equals the cipher of the product.

$Enc(x . y) = Enc(x).Enc(y)$.

4.2.3.4 Additive and Multiplicative Homomorphic Encryption Systems

These systems allow arbitrary many homomorphic computations of one type and limited number of operations of the other type i.e. it allows both addition and multiplication operations but is not fully homomorphic. An example of this kind would be Boneh-Goh-Nissim cryptosystem . It supports computation of an unlimited number of additions but at most one multiplication.

4.2.3.4 Fully Homomorphic Encryption Systems

These systems allow arbitrary number of additions and multiplications and thus many types of computations can be done on the encrypted data that is stored in the cloud without the need for any decryption. This means that operations on confidential data can now be outsourced to the cloud server keeping the secret key that can decrypt the result of the operation.{Mohammad, 2014 #10}

4.2.4 Homomorphic Encryption Applied to Cloud Computing Security:

When the data transferred to the Cloud we use standard encryption methods to secure the operations and the storage of the data. Our basic concept was to encrypt the data before send it to the Cloud provider. But the last one needs to decrypt data at every operation. The client will need to provide the private key to the server (Cloud provider) to decrypt data before execute the calculations required, which might affect the confidentiality and privacy of data stored in the Cloud. In this paper we are proposing an application of a method to execute operations on encrypted data without decrypting them, which will provide the same results after calculations as if we have worked directly on the raw data.{Tebaa, 2012 #15}

4.2.5 Paillier cryptosystem

The Paillier Cryptosystem named after and invented by French researcher Pascal Paillier in 1999 is an algorithm for public key cryptography.

The distinguishing technique used in public key cryptography is the use of asymmetric key algorithms, where the key used to encrypt a message is not the same as the key used to decrypt it. Each user has a pair of cryptographic keys , a public key and a private key. The private key is kept secret, whilst the public key may be widely distributed. Messages are encrypted with the recipient's public key and can only be decrypted with the corresponding private key. The keys are related mathematically, but the private key cannot be feasibly derived from the public key. [16]

Key generation

1. Choose two large prime numbers *p* and *q* randomly and independently of each other such that $\gcd(pq,(p-1)(q-1))=1$. This property is assured if both primes are of equal length
2. Compute $n=pq$and $\lambda=\operatorname{lcm}(p-1,q-1)$.
3. Select random integer $g$where $g\in\mathbb{Z}_{n^2}^*$
4. Ensure $n$divides the order of $g$by checking the existence of the following modular multiplicative inverse: $\mu=(L(g^\lambda \bmod n^2))^{-1} \bmod n$,

   where function L is defined as $L(u)=\dfrac{u-1}{n}$.

   Note that the notation $\dfrac{a}{b}$does not denote the modular multiplication of a times the modular multiplicative inverse of bb ut rather the quotient of a divided by b, i.e., the largest integer value $v\geq 0$to satisfy the relation $a\geq vb$.

- The public (encryption) key is (n,g).
- The private (decryption) key is (λ,μ).

If using p,q of equivalent length, a simpler variant of the above key generation steps would be to set $g=n+1, \lambda=\varphi(n)$,and $\mu=\varphi(n)^{-1} \bmod n$, where $\varphi(n)=(p-1)(q-1)$

Encryption

1. Let m be a message to be encrypted where $m \in \mathbb{Z}_n$
2. Select random r where $r \in \mathbb{Z}_n^*$
3. Compute ciphertext as: $c = g^m \cdot r^n \bmod n^2$

Decryption

1. Let c be the ciphertext to decrypt, where $c \in \mathbb{Z}_{n^2}^*$
2. Compute the plaintext message as: $m = L(c^\lambda \bmod n^2) \cdot \mu \bmod n$

Homomorphic properties

A notable feature of the Paillier cryptosystem is its homomorphic properties along with its non-deterministic encryption . As the encryption function is additively homomorphic, the following identities can be described:

- **Homomorphic addition of plaintexts**

  The product of two cipher texts will decrypt to the sum of their corresponding plaintexts,
  $D(E(m_1, r_1) \cdot E(m_2, r_2) \bmod n^2) = m_1 + m_2 \bmod n.$

  The product of a ciphertext with a plaintext raising g will decrypt to the sum of the corresponding plaintexts,

  $D(E(m_1, r_1) \cdot g^{m_2} \bmod n^2) = m_1 + m_2 \bmod n.$

- **Homomorphic multiplication of plaintexts**

  An encrypted plaintext raised to the power of another plaintext will decrypt to the product of the two plaintexts,
  $D(E(m_1, r_1)^{m_2} \bmod n^2) = m_1 m_2 \bmod n,$

  $D(E(m_2, r_2)^{m_1} \bmod n^2) = m_1 m_2 \bmod n.$

  More generally, an encrypted plaintext raised to a constant *k* will decrypt to the product of the plaintext and the constant,

  $D(E(m_1, r_1)^k \bmod n^2) = k m_1 \bmod n.$

5.1 INTRODUCTION:

This chapter shows the implementation steps and screen shots for the basic operation.

5.2 IMPLEMENTATION STEPS:

5.2.1 Cloud Server:

The cloud software used is OpenStack and its components were configured in this project and that mentioned in details at APPENDIX II. and here the admin and client interfaces to interact with the cloud server:

5.2.1.1 Login Interface:

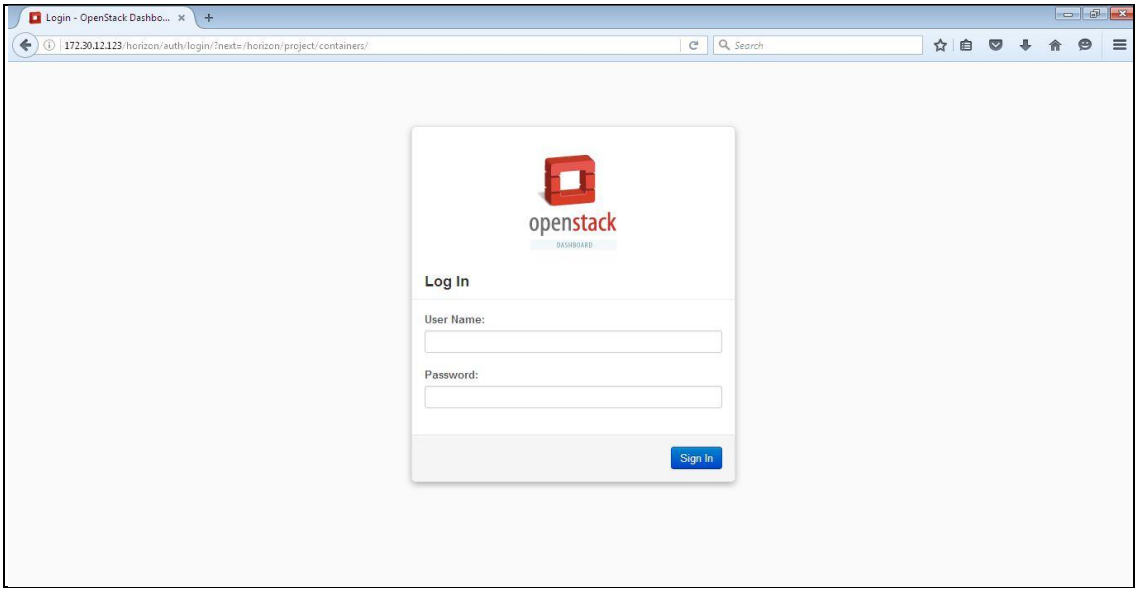

Figure 5.1 Login Interface.

5.2.1.2 Users Account:

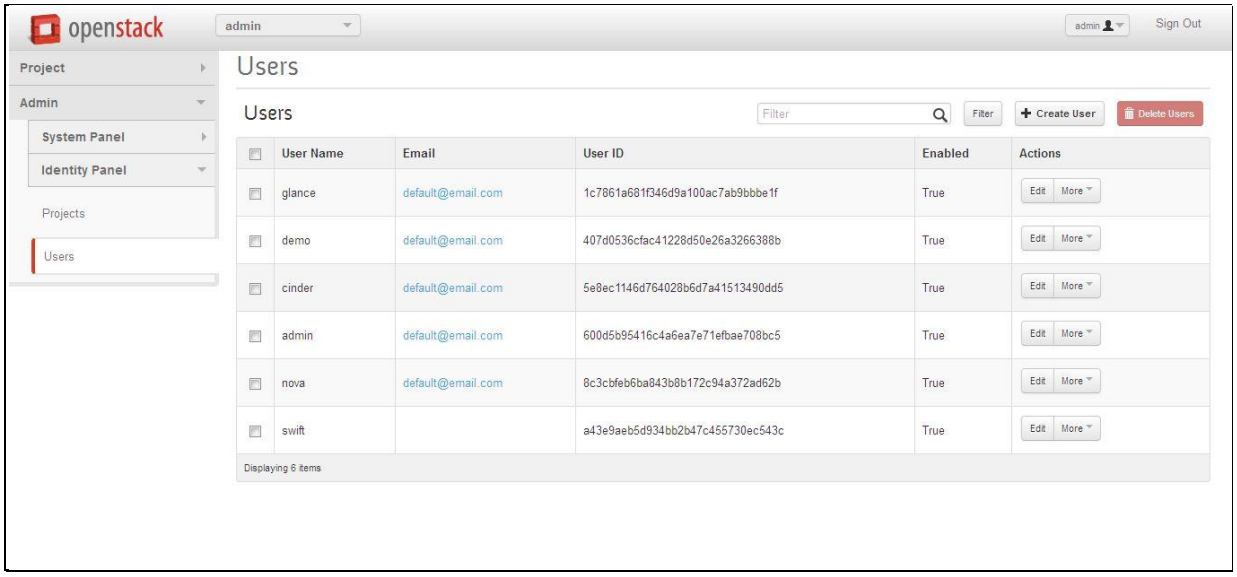

Figure 5.2Users Accounts Interface.

5.2.1.3 Admin Panel Interface:

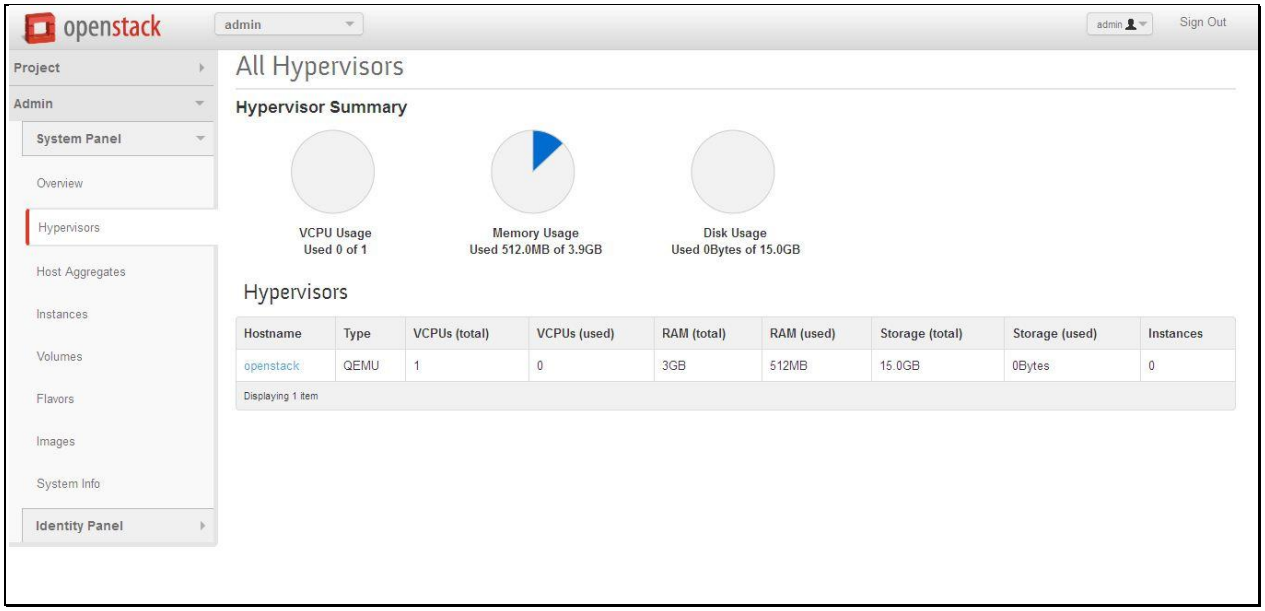

Figure 2.3: Admin Interface.

5.2.1.4 Containers Interface:

Container play a folder role for the clients they can organize their files in different containers or put all files in one container.

Figure 5.4: Shows The Containers Interface.

5.2.1.5 Client Objects Interface:

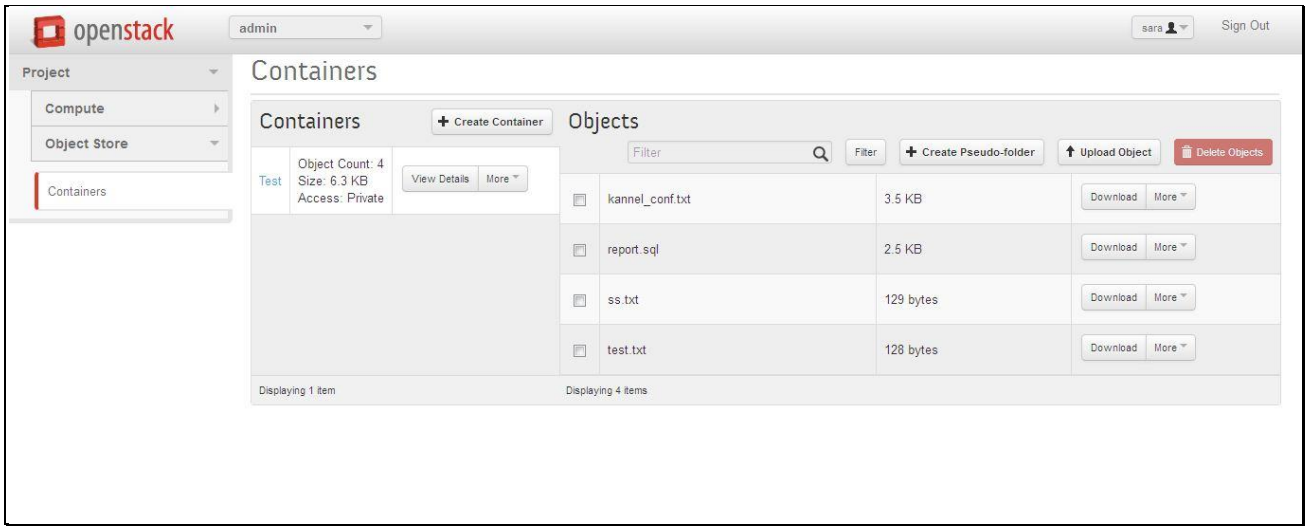

Figure 5.5: Shows The Objects Interface.

5.2.2 Web Based Application :

The application was developed using java EE (JSP and java servlet) it communicate with OpenStack API to provide the desired services, the communication with API was done through the HTTP protocol on secure port 8080 any request associate with account credentials and Authorization token.

The application provide security using paillier algorithm as one of the Homomorphic encryption algorithms.

5.2.2.1 **Login :**

User login using the appropriate credentials(username ,password) based on his account ,Then its verified using the authentication process in OpenStack server . In response to

the credentials, the identity service issues an authentication token that the user must provides for subsequent requests.

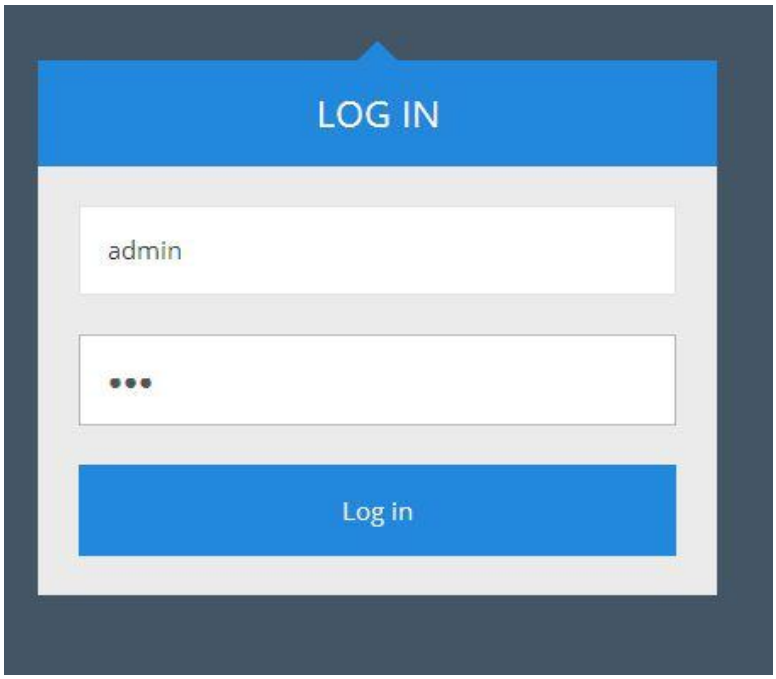

Figure 5.6: Shows The Login Interface.

5.2.2.2 Home Page :

This interface consists of various functionalities such as uploading of files to the cloud that include encryption mechanism, download mechanism and decryption mechanism to view the decrypted files.

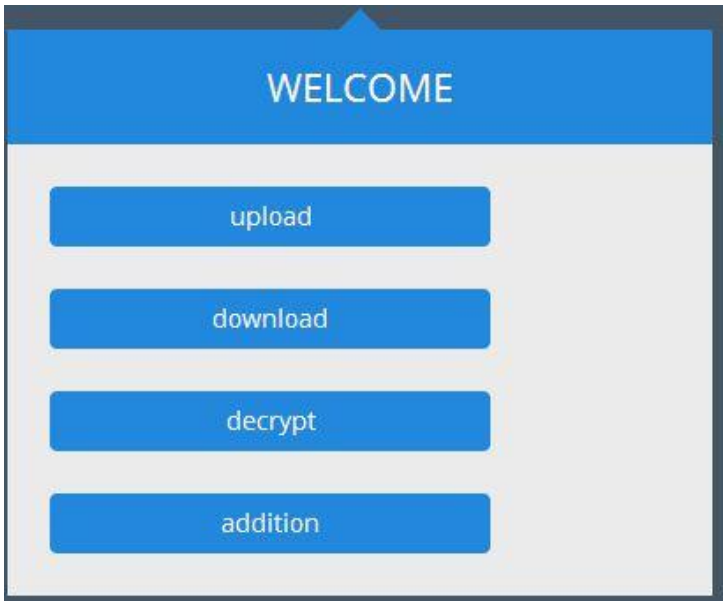

Figure 5.7: Shows The User Interface.

5.2.2.3 File Upload Mechanism

Using this mechanism user can upload data onto the website in encrypted format .this mechanism include the encryption module to protect the privacy and the two keys stored locally.

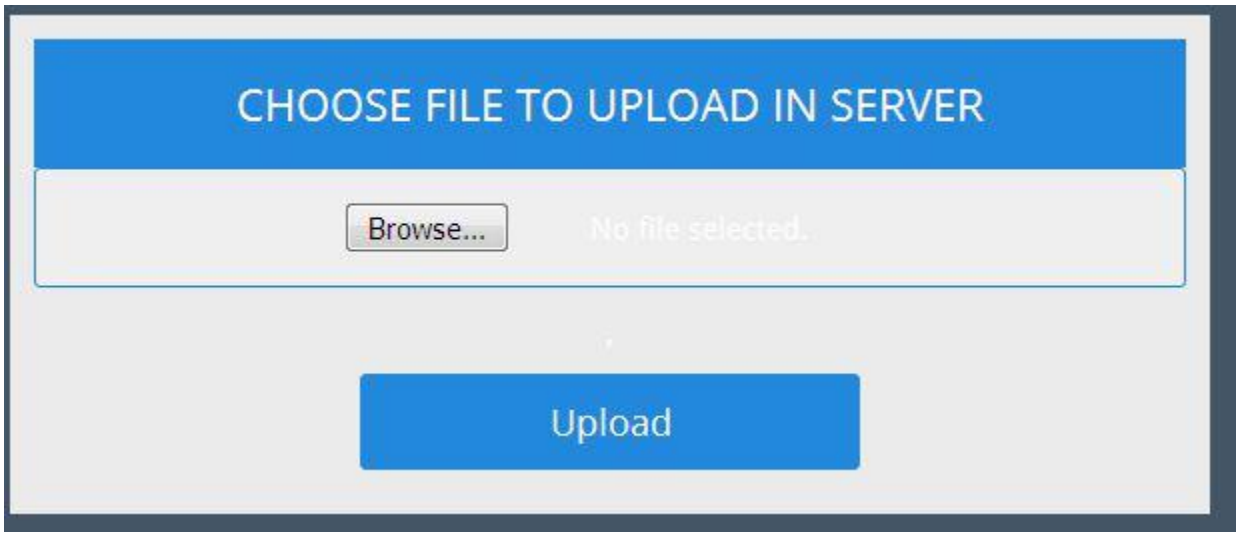

Figure 5.8:The Upload Interface.

5.2.2.4 File Encryption Module:

This module doesn't have user interface it is an embedded module it take the file that wanted to upload and convert it to byte array and divide it to sub arrays each one has length is 64 byte , then convert each one to Big Integer and passed as plain text to paillier encryption method that return cipher text in Big Integer format and it convert again to byte array.

5.2.2.5 Download Encrypted File:

user can download the encrypted file ,this include two steps first list all files related to the user account , user select the file want to download it .

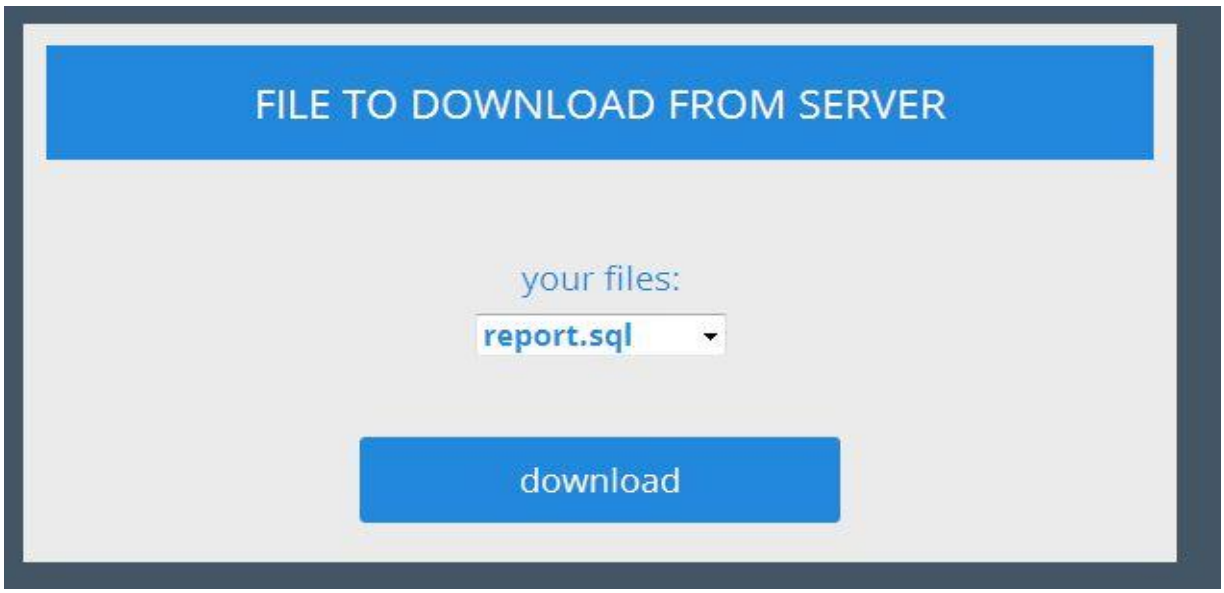

Figure 5.9: The download interface.

5.2.2.6 Decryption Mechanism:

After user download encrypted file he/she can decrypt it because the private key stored locally , the selected file convert to byte array and divide it to sub arrays each one has length is 128 byte , then convert each one to Big Integer and passed as cipher text to paillier Decryption method that return plain text in Big Integer format and it convert again to byte array.

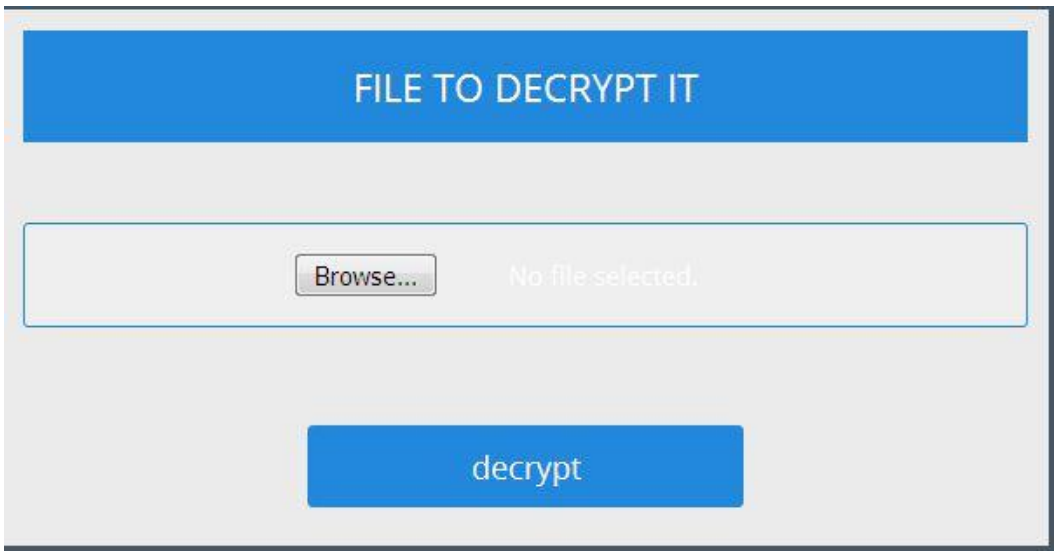

Figure 5.10: The Decryption Interface.

5.2.2.7 Homomorphic addition Mechanism:

After user upload encrypted file he/she can apply addition operation on the encrypted data without download it ,by send encrypted value you want to add and select the file convert to byte array and , the result will re-write in encrypted format .

This operation was done in secure way and without decrees the performance.

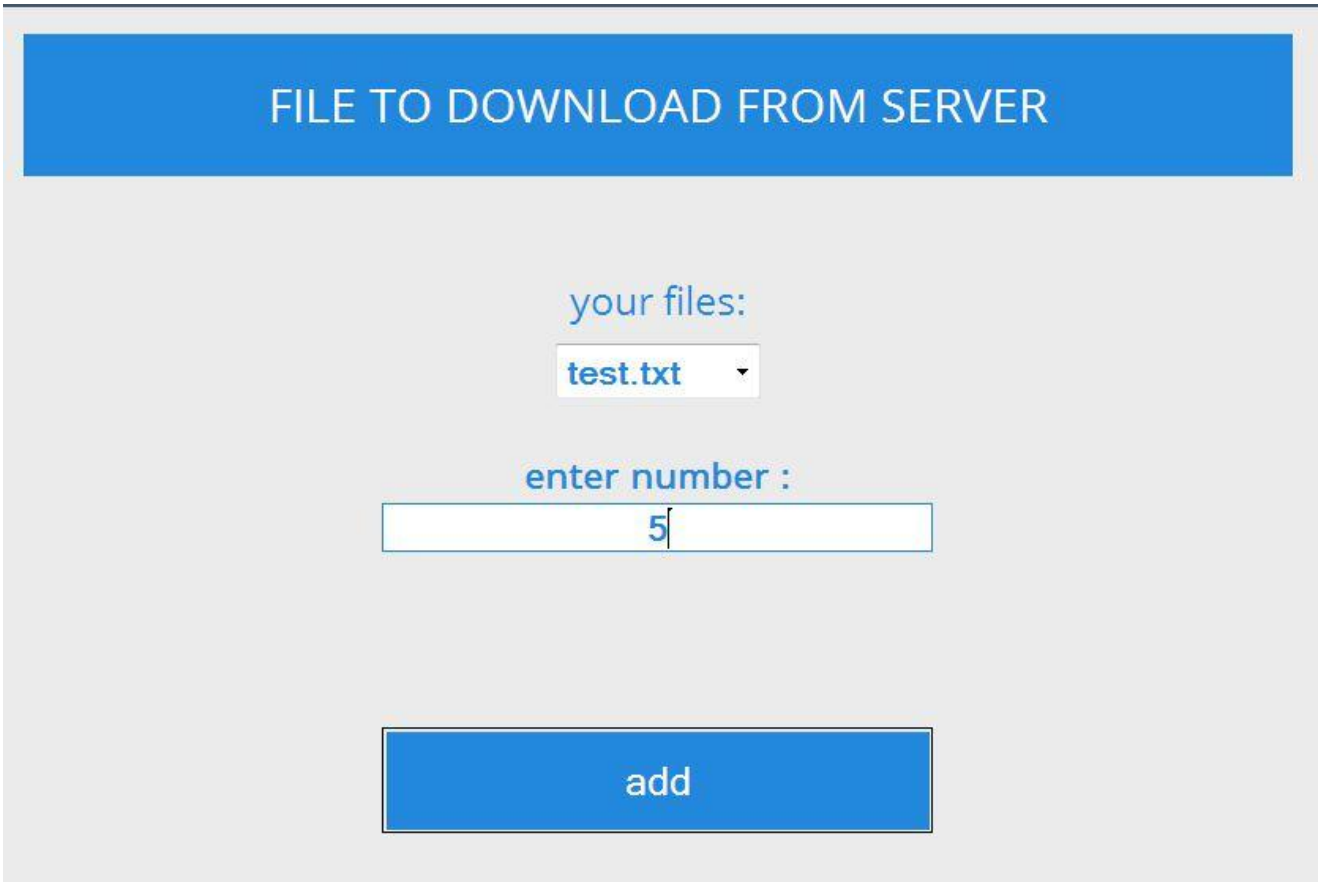

Figure 5.11: The Addition Interface.

5.3Discussions :

- The major challenges that any cloud computing service provider must overcome are to make sure that if its servers are attacked by hackers, the client data cannot be stolen or misused and this was achieved by encrypt the client data before upload it .

- The confidential client data must remain invisible even to the cloud service providers. this was achieved by The client private key store locally and decryption done on the client side.
- The provider can operate on the data without decrypt it . this was achieved by using Homomorphic encryption.

5.4 Results:

This implementation of a cloud integrated with web application can provide functionality for users to homomorphically encrypt data on the cloud servers. The main advantage of this technique is that operations such as addition can be done on encrypted data without the need for decryption. The user's data is kept confidentiality as the cloud server that operates on it does not know what data it operated upon. Also, if the cloud service provider servers are hacked by malicious attackers, the user's data is secured and cannot be misused as it is homomorphic encrypted.

Sample of results:

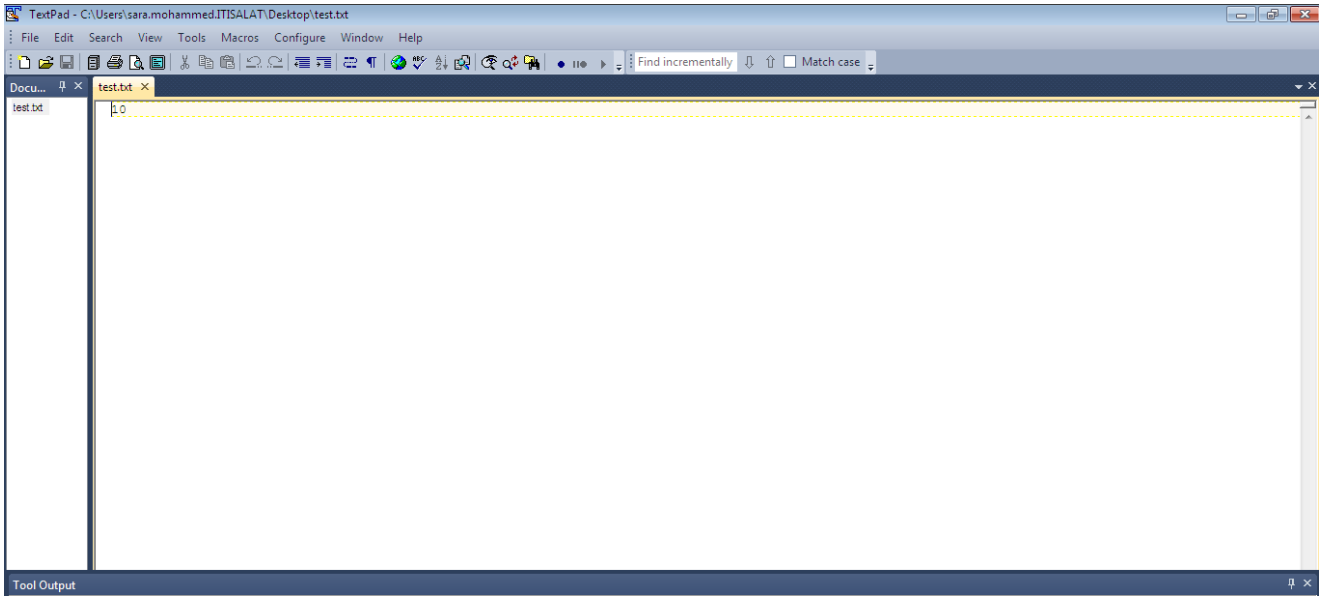

Figure 5.12: The plain text file.

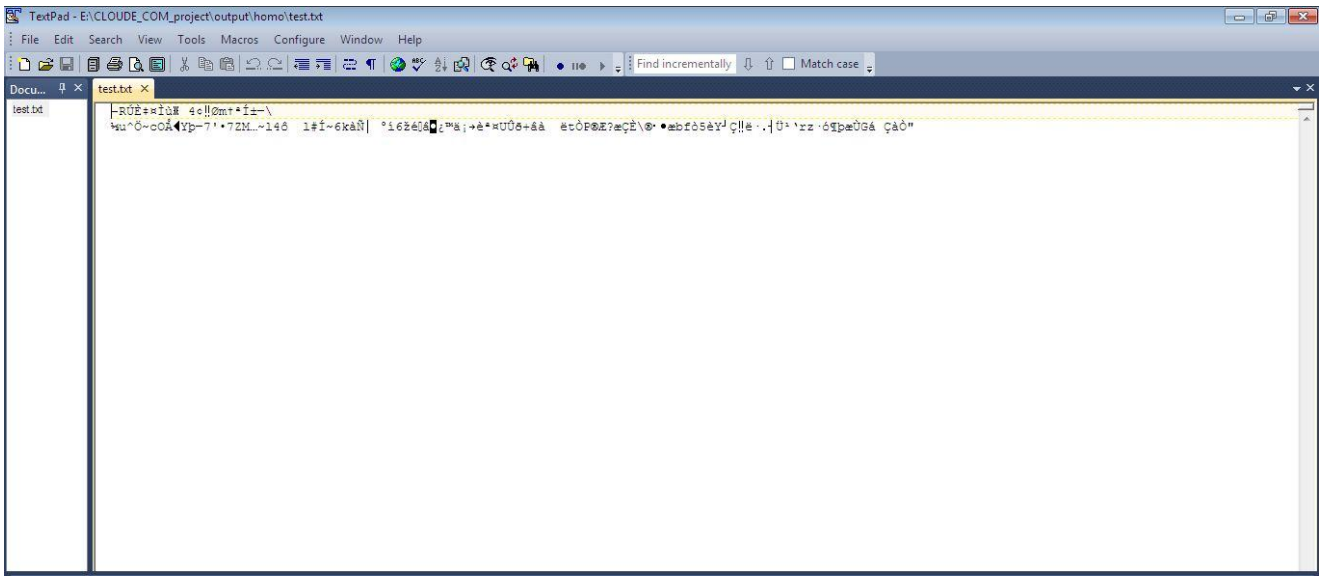

Figure 5.13: The encrypted text file after downloaded.

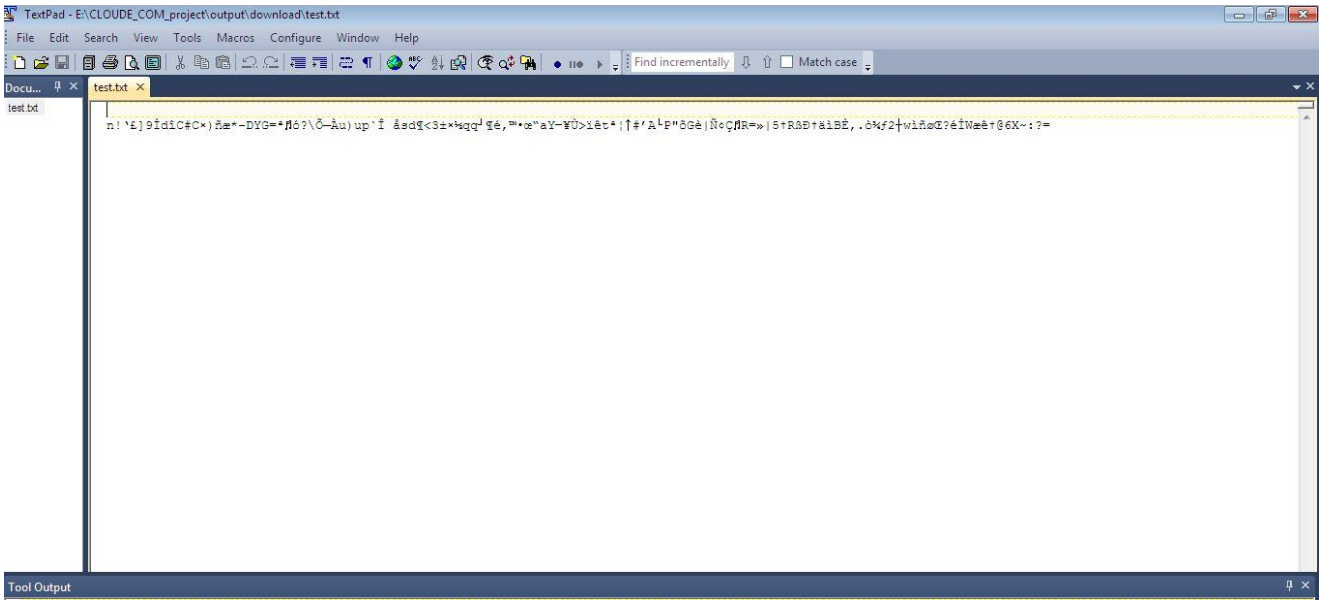

Figure 5.14: The encrypted modified text file after downloaded.

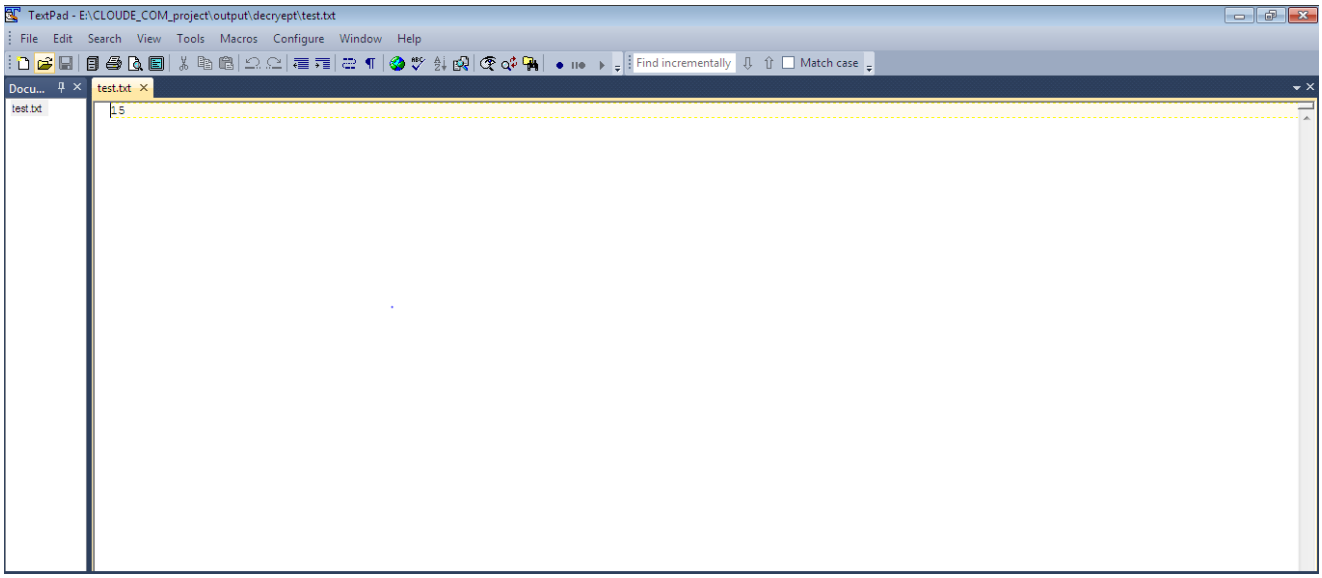

Figure 5.15: The decrypted modified text file.

6.1 CONCLUSION

In this thesis a cloud based web application was implemented to encrypt data on the cloud servers using Paillier's homomorphic encryption algorithm. The main advantage of this technique is that operations can be performed on encrypted data stored on the cloud without the need for decryption. Hence, user's data is kept confidentiality as the cloud server that operates on it does not know what data it operated upon. Also, if the cloud service provider servers are hacked by malicious attackers, the user's data is secured and cannot be misused as it is homomorphic encrypted.

Homomorphic encryption in cloud computing is an active area of research considering the dominance of cloud computing in today's market place. therefore more work can be done in future to build more efficient cloud computing applications similar to the one that is implemented as part of this project. Additional functionality to work with different types of input files can be implemented for this application in the future.

6.2 RECOMMENDATION

I recommends with the following:

- Use the Fully Homomorphic Encryption rather than Additive Homomorphic Encryption to include all possible operations of the data.
- Apply this solution into cloud Data base service (Daas) .
- Work in others security service (integrity and Availability).

6.3 OBSTACLES:

The implementation of this research faced many problems:

- Ubuntu12.04 was used to build OpenStack from the source which was failed many times because the deeps files were not found. Then Migrate to Ubuntu14.04, the setup of all OpenStack components was successfully installed.
- Fully Homomorphic Encryption(FHE) java package was used at the beginning but the integration with Java servlet failed ,then migrate to Additive Homomorphic.

REFERENCES:

{Sarga, 2012 #1}Sarga, L. (2012). Cloud computing: an overview. Journal of Systems Integration, 3(4), 3.[4]

{Maddineni,2012#2} Maddineni, V. S. K., & Ragi, S. (2012). Security Techniques for protecting data in Cloud Computing.

{ Annapureddy ,2010#3}Annapureddy, K. (2010). Security challenges in hybrid cloud infrastructures. Aalto University.

{#4} https://www.cloudsocket.eu/common-understanding-wiki/-/wiki/Main/Cloud+Actors [Access on 3-2-2016]

{ASHALATHA, #5}ASHALATHA, R., & VAIDEHI, M. THE SIGNIFICANCE OF DATA SECURITY IN CLOUD: A SURVEY ON CHALLENGES AND SOLUTIONS ON DATA SECURITY.

{Jiménez Martínez, 2013 #6} Jiménez Martínez, D. (2013). Privacy and confidentiality issues in cloud computing architectures.

{Gampala, 2012 #7} Gampala, V., Inuganti, S., & Muppidi, S. (2012). Data security in cloud computing with elliptic curve cryptography. International Journal of Soft Computing and Engineering (IJSCE), 2(3), 138-141.

{Sarddar, 2015 #8} Sarddar, D., Das, N., & Halder, J. (2015). An Authenticate Model of Cloud Interaction Using Cryptography. International Journal of Grid and Distributed Computing, 8(6), 9-18.

{Mohammad, 2014 #9} Mohammad, M. K. (2014). Using Homomorphic Encryption to Protect Confidentiality and Integrity of Data in a Simulated Cloud Environment. Texas A&M University-Corpus Christi.

{#10} https://www.ubuntu.com/about/about-ubuntu [Access on 5-3-2016]

{#11} OpenStack Installation Guide for Ubuntu 12.04/14.04 (LTS), icehouse (june 1,2015)

{#12} http://docs.openstack.org/openstack-ops/content/storage_decision.html [Accessed on 2-13-2016]

{#13} http://www.pcmag.com/encyclopedia/term/57268/java-ee [Accessed on 15-3-2016]

{#14} http://zeroturnaround.com/rebellabs/using-eclipse-for-java-development/ [Accessed on 15-3-2016]

{Tebaa, 2012 #15}Tebaa, M., El Hajji, S., & El Ghazi, A. (2012). Homomorphic encryption applied to the cloud computing security. Paper presented at the Proceedings of the World Congress on Engineering.

{#16} http://security.hsr.ch/msevote/seminarpapers/HS09_Homomorphic_Tallying_with_Paillier.pdf[Accessed on 20-3-2016]

APPENDIX:

APPENDIX I:

Paillier's Algorithm:

The following code implements Paillier's Algorithm for integer and text data.

```
//package Paillier;

/**

 * @author Sara

 */

/**

 * This program is free software: you can redistribute it and/or modify it under the terms of the GNU
* General Public License as published by the Free Software Foundation, either version 3 of the
*License, or (at your option)

 */

Import java.math.*;

Import java.util.*;

/**

 * Paillier Cryptosystem <br><br>

 * References: <br>

 * [1] Pascal Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes,"
EUROCRYPT'99.

 * URL: <a
href="http://www.gemplus.com/smart/rd/publications/pdf/Pai99pai.pdf">http://www.gemplus.co
m/smart/rd/publications/pdf/Pai99pai.pdf</a><br>

* [2] Paillier cryptosystem from Wikipedia.
```

```
 *                                                    URL:                    <a
href="http://en.wikipedia.org/wiki/Paillier_cryptosystem">http://en.wikipedia.org/wiki/Paillier_cryp
tosystem</a>

 * @author Kun Liu (kunliu1@cs.umbc.edu)

 * @version 1.0

 */

public class Paillier {

   /**

    * p and q are two large primes.

    * lambda = lcm(p-1, q-1) = (p-1)*(q-1)/gcd(p-1, q-1).

    */

Private BigInteger p,  q,  lambda;

   /**

    * n = p*q, where p and q are two large primes.

    */

Public BigInteger n;

   /**

    * nsquare = n*n

    */

Public BigInteger nsquare;

   /**

    * a random integer in Z*_{n^2} where gcd (L(g^lambda mod n^2), n) = 1.

    */

Private BigInteger g;

   /**

    * number of bits of modulus
```

```
   */
Private int bitLength;
   /**
    * Constructs an instance of the Paillier cryptosystem.
    * @param bitLengthVal number of bits of modulus
    * @param certainty The probability that the new BigInteger represents a prime number will exceed (1 - 2^(-certainty)). The execution time of this constructor is proportional to the value of this parameter.
    */
Public Paillier(int bitLengthVal, int certainty) {
KeyGeneration(bitLengthVal, certainty);
   }
   /**
    * Constructs an instance of the Paillier cryptosystem with 512 bits of modulus and at least 1-2^(-64) certainty of primes generation.
    */
public Paillier() {
KeyGeneration(512, 64);
   }
   /**
    * Sets up the public key and private key.
* @param bitLengthVal number of bits of modulus.
    * @param certainty The probability that the new BigInteger represents a prime number will exceed (1 - 2^(-certainty)). The execution time of this constructor is proportional to the value of this parameter.   */
public void KeyGeneration(int bitLengthVal, int certainty) {
bitLength = bitLengthVal;
```

```
    /*Constructs two randomly generated positive BigIntegers that are probably prime, with the specified bitLength and certainty.*/

    p = new BigInteger(bitLength / 2, certainty, new Random());

    q = new BigInteger(bitLength / 2, certainty, new Random());

    n = p.multiply(q);

nsquare = n.multiply(n);

    g = new BigInteger("2");

lambda = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE)).divide(

p.subtract(BigInteger.ONE).gcd(q.subtract(BigInteger.ONE)));

    /* check whether g is good.*/

if (g.modPow(lambda, nsquare).subtract(BigInteger.ONE).divide(n).gcd(n).intValue() != 1) {

System.out.println("g is not good. Choose g again.");

System.exit(1);

    }

  }

  /**

   * Encrypts plaintext m. ciphertext c = g^m * r^n mod n^2. This function explicitly requires random input r to help with encryption.

   * @param m plaintext as a BigInteger

   * @param r random plaintext to help with encryption

   * @return ciphertext as a BigInteger

   */

Public BigInteger Encryption(BigInteger m, BigInteger r) {

return g.modPow(m, nsquare).multiply(r.modPow(n, nsquare)).mod(nsquare);

  }

  /**
```

```
    * Encrypts plaintext m. ciphertext c = g^m * r^n mod n^2. This function automatically generates random input r (to help with encryption).

    * @param m plaintext as a BigInteger

    * @return ciphertext as a BigInteger

    */

Public BigInteger Encryption(BigInteger m) {

BigInteger r = new BigInteger(bitLength, new Random());

Return g.modPow(m, nsquare).multiply(r.modPow(n, nsquare)).mod(nsquare);

  }

  /**

    * Decrypts ciphertext c. plaintext m = L(c^lambda mod n^2) * u mod n, where u = (L(g^lambda mod n^2))^(-1) mod n.

    * @param c ciphertext as a BigInteger

    * @return plaintext as a BigInteger

    */

Public BigInteger Decryption(BigInteger c) {

BigInteger u = g.modPow(lambda, nsquare).subtract(BigInteger.ONE).divide(n).modInverse(n);

Return c.modPow(lambda, nsquare).subtract(BigInteger.ONE).divide(n).multiply(u).mod(n);

  }

  /**

My own functions to manipulate  Big Integer

    */

public static BigInteger toBigInteger(String s){return new BigInteger(s.getBytes());}

public static String fromBigInteger(BigInteger b){return new String(b.toByteArray());}

}
```

APPENDIX II:

Installing OpenStack Icehouse on Ubuntu 14.04 LTS:

1- fresh install of Ubuntu 14.04 LTS Desktop, you'll need to locally login to each rig and install the *openssh-server* to allow remote *ssh* access:

```
sudo apt-get install openssh-server
```

2- Remotely log into your new server and install *git* with *aptitude*:

```
Sudo su
apt-get -y install git
```

3- Checkout the StackGeekOpenStack setup scripts from Github:

```
git clone git://github.com/StackGeek/openstackgeek.git
cd openstackgeek/icehouse
```

4- Setup

Note: Be sure to take a look at the scripts before you run them. Keep in mind the setup scripts will periodically prompt you for input, either for confirming installation of a package, or asking you for information for configuration. Start the installation by running the setup script:

```
./openstack_setup.sh
```

5- Database Setup :

The next part of the setup installs MySQL and RabbitMQ.

```
./openstack_mysql.sh
```

The install script will install Rabbit and MySQL. During the MySQL install you will be prompted for the MySQL password you entered earlier to set a password for the MySQL root user. You'll be prompted again toward the end of the script when it creates the databases.

6- Keystone Setup :

Keystone is used by OpenStack to provide central authentication across all installed services. Start the install of Keystone by typing the following:

```
./openstack_keystone.sh
```

When the install is done, test Keystone by setting the environment variables using the newly created **stackrc** file.

```
. ./stackrc
keystone user-list
```

Keystone should output the current user list to the console:

```
+----------------------------------+---------+---------+--------------------+
|                id                |   name  | enabled |       email        |
+----------------------------------+---------+---------+--------------------+
| 5474c43e65c840b5b371d695af72cba4 |  admin  |   True  | xxxxxxxx@gmail.com |
| dec9e0adf6af4066810b922035f24edf |  cinder |   True  | xxxxxxxx@gmail.com |
| 936e0e930553423b957d1983d0a29a62 |   demo  |   True  | xxxxxxxx@gmail.com |
| 665bc14a5da44e86bd5856c6a22866fb |  glance |   True  | xxxxxxxx@gmail.com |
| bf435eb480f643058e27520ee3737685 |   nova  |   True  | xxxxxxxx@gmail.com |
| 7fa480363a364d539278613aa7e32875 | quantum |   True  | xxxxxxxx@gmail.com |
+----------------------------------+---------+---------+--------------------+
```

7- Glance Setup :

Glance provides image services for OpenStack. Images are comprised of prebuilt operating system images built to run on OpenStack. Start the Glance install by typing:

```
./openstack_glance.sh
```

Once the Glance install completes, you should be able to query the system for the available images:

```
glance image-list
```

The output should be something like this:

```
+--------------------------------------+--------------+-------------+--------+----
| ID                                   | Name         | Disk Format | Format |
Size     | Status |
+--------------------------------------+--------------+-------------+--------+----
```

```
| df53bace-b5a0-49ba-9b7f-4d43f249e3f3 | Cirros 0.3.0 | qcow2         | bare    |
9761280 | active |
+--------------------------------------+-------------+------------+--------+----
```

8- Cinder Setup :

Cinder is used to provide additional volume attachments to running instances and snapshot space. Start the install of Cinder by typing:

```
./openstack_cinder.sh
```

Once the install of Cinder is complete, determine your space requirements and run the loopback volume creation script (keep in mind you have to create a loopback file that is at least 1GB in size):

```
./openstack_loop.sh
```

You should now be able to query installed storage types:

```
cinder type-list
```

You may then create a new volume to test:

```
cinder create --volume-type Storage --display-name test 1
```

9- Nova Setup :

Nova provides multiple services to OpenStack for controlling networking, imaging and starting and stopping instances. Start the controller's nova install by typing the following:

```
./openstack_nova.sh
```

When the install is complete, you may query the running services by doing the following:

```
nova service-list
```

You should see output that looks similar to this:

```
+------------------+--------+----------+---------+-------+------------------------
| Binary               | Host     | Zone        | Status    | State | Updated_at
|
+------------------+--------+----------+---------+-------+------------------------
| nova-cert             | tester  | internal  | enabled  | up       | 2014-02-
20T10:37:25.000000 |
| nova-conductor      | tester  | internal  | enabled  | up       | 2014-02-
20T10:37:17.000000 |
```

```
|  nova-consoleauth  |  tester  |  internal  |  enabled  |  up        |  2014-02-
20T10:37:25.000000 |
|  nova-network        |  tester  |  internal  |  enabled  |  up        |  2014-02-
20T10:37:25.000000 |
|  nova-scheduler      |  tester  |  internal  |  enabled  |  up        |  2014-02-
20T10:37:24.000000 |
+------------------+--------+----------+---------+-------+----------------------
```

10- Swift setup:

```
./openstack_swift_loop.sh
```

11- Nova Compute Setup

```
./openstack_nova_compute.sh

nova service-list
```

You should see new entries for the newly added compute rig:

```
+------------------+---------+----------+---------+-------+----------------------
|  Binary              |  Host      |  Zone       |  Status   |  State  |  Updated_at
|
+------------------+---------+----------+---------+-------+----------------------
|  nova-cert             |  nero      |  internal  |  enabled  |  up       |  2014-04-
13T17:20:52.000000 |
|  nova-compute        |  booster  |  nova        |  enabled  |  up       |  2014-04-
13T17:20:55.000000 |
|  nova-compute        |  nero      |  nova        |  enabled  |  up       |  2014-04-
13T17:20:55.000000 |
|  nova-conductor     |  nero       |  internal  |  enabled  |  up       |  2014-04-
13T17:20:52.000000 |
|  nova-consoleauth  |  nero       |  internal  |  enabled  |  up       |  2014-04-
13T17:20:52.000000 |
|  nova-network        |  booster  |  internal  |  enabled  |  up       |  2014-04-
13T17:20:52.000000 |
|  nova-network        |  nero       |  internal  |  enabled  |  up       |  2014-04-
13T17:20:52.000000 |
|  nova-scheduler     |  nero       |  internal  |  enabled  |  up       |  2014-04-
13T17:20:52.000000 |
+------------------+---------+----------+---------+-------+----------------------
```

12- Horizon Setup :

Horizon provides OpenStack's managment interface. Install Horizon by typing:

```
./openstack_horizon.sh
reboot
```

you should be able to log into your OpenStack cluster with the following URL format (changing the IP of course):

```
http://172.30.12.123/horizon
```