

## **A Novel Proxy Re-Encryption Technique for Secure Data Sharing in Cloud Environment**

### **BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE & ENGINEERING - CYBER SECURITY**

**Submitted by**

<b>B PREETHI</b>	<b>20WJ1A6249</b>
<b>B ANUSHA</b>	<b>21WJ1A6211</b>
<b>K ABHINAV KRISHNA</b>	<b>21WJ1A6230</b>
<b>P VINAYA BASWARAJ</b>	<b>21WJ1A6250</b>

*Under the Guidance of*

**Mr. Ch. Upendar, M.Tech(Ph.D)**  
**Assistant Professor in CSE-Cyber Security**

**Department of CSE-Cyber Security**  
**School of Engineering and Technology**  
**Guru Nanak Institutions Technical Campus**  
**Ibrahimpattanam, Hyderabad, R.R. District – 501506**



## **ABSTRACT**

This project implements a Proxy Re-Encryption (PRE) system for secure data sharing and management in a cloud-based environment. The core objective is to enable a cloud owner to securely upload data, divide it into multiple parts, and allow a data user to access and modify this data while maintaining confidentiality and integrity. Initially, the cloud owner uploads a file, which is split into four segments. Each segment undergoes a hashing process using the SHA algorithm, generating four distinct SHA hash values, which serve as integrity verifiers for the respective parts of the data. To protect data during the sharing process, the owner encrypts the data before granting access to a specific data user. The encrypted file is then subjected to proxy re-encryption, where a proxy server, without learning anything about the contents of the file, re-encrypts it on behalf of the owner. This allows the data user to decrypt the file securely with the provided key. The cloud server manages this process by performing the re-encryption and sharing the encrypted file and key with the user. In this system, only authorized users with the necessary permissions, granted by the cloud owner, can access the file content. The use of proxy re-encryption ensures that the cloud server does not need to have direct access to the unencrypted data, preserving data privacy and security. This project enhances secure data sharing and access in cloud environments, offering a high level of data confidentiality, integrity, and control for the owner while enabling safe and selective access for users.



## LIST OF FIGURES


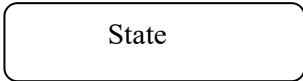
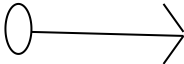
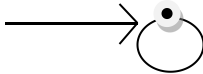
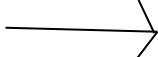
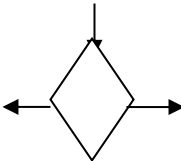
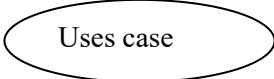
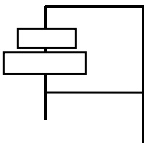
FIGURE	TITLE	PAGE NUMBER
3.2.3	Proposed model	10
3.3.1.1	User interface design	11
3.3.1.2	Admin	12
3.3.1.3	Cloud owner	13
3.3.1.4	Cloud proxy server	14
3.3.1.5	Data user	15
3.4.1	Use case diagram	19
3.4.2	Class diagram	20
3.4.3	Object diagram	21
3.4.4	Component diagram	22
3.4.5	Deployment diagram	23
3.4.6	Sequence diagram	24
3.4.7	Collaboration diagram	25
3.4.8	State diagram	26
3.4.9	Activity diagram	27
3.4.10	Data flow diagram	28



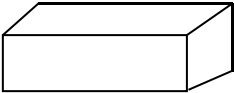
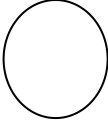


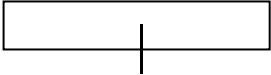

# LIST OF SYMBOLS

S.NO	NAME	NOTATION	DESCRIPTION
1.	Class	<div> <div>+ public -private # protected</div> <div> <div>Class Name</div> <div>-attribute -attribute</div> <div>+operation +operation +operation</div> </div> </div>	Represents a collection of similar entities grouped together.
2.	Association	<div> <div>Class A</div> <div>Class B</div> </div> <div> <div>Class A</div> <div>Class B</div> </div>	Associations represents static relationships between classes. Roles represents the way the two classes see each other.
3.	Actor		It aggregates several classes into a single classes.
4.	Aggregation	<div> <div>Class A</div> <div>Class B</div> </div> <div> <div>Class A</div> <div>Class B</div> </div>	Interaction between the system and external environment
5.	Relation (uses)	Uses	Used for additional process communication.
6.	Relation (extends)	<div>extends</div> <div>→</div>	Extends relationship is used when one use case is similar to another use case but does a bit more.



7.	Communication		Communication between various use cases.
8.	State		State of the process.
9.	Initial State		Initial state of the object
10.	Final state		Final state of the object
11.	Control flow		Represents various control flow between the states.
12.	Decision box		Represents decision making process from a constraint
13.	Usecase		Interact ion between the system and external environment.
14.	Component		Represents physical modules which is a collection of components.



15.	Node		Represents physical modules which are a collection of components.
16.	Data Process/State		A circle in DFD represents a state or process which has been triggered due to some event or action.
17.	External entity		Represents external entities such as keyboard, sensors, etc.
18.	Transition		Represents communication that occurs between processes.
19.	Object Lifeline		Represents the vertical dimensions that the object communications.
20.	Message	Message 	Represents the message exchanged.



## CHAPTER 1

### INTRODUCTION

#### 1.1 General

The Internet of Things (IoT) has emerged as a technology that has great significance to the world nowadays and its utilization has given rise to an expanded growth in network traffic volumes over the years. It is expected that a lot of devices will get connected in the years ahead. Data is a central notion to the IoT paradigm as the data collected serves several purposes. in applications such as healthcare, vehicular networks, smart cities, industries, and manufacturing, among others. The sensors measure a host of parameters that are very useful for stakeholders involved. Consequently, as enticing as IoT seems to be, its advancement has introduced new challenges to security and privacy. IoT needs to be secured against attacks that hinder it from providing the required services, in addition to those that pose threats to the confidentiality, integrity, and privacy of data. A viable solution is to encrypt the data before outsourcing to the cloud servers. Attackers can only see the data in its encrypted form when traditional security measures fail. In data sharing, any information must be encrypted from the source and only decrypted by authorized users in order to preserve its protection. Conventional encryption techniques can be used, where the decryption key is shared among all the data users designated by the data owner. The use of symmetric encryption implies that the same key is shared between the data owner and users, or at least the participants agree on a key. This solution is very inefficient. Furthermore, the data owners do not know in advance who the intended data users are, and, therefore, the encrypted data needs to be decrypted and subsequently encrypted with a key known to both the data owner and the users. This decrypt-and-encrypt solution means the data owner has to be online all the time, which is practically not feasible. The problem becomes increasingly complex when there are multiple pieces of data and diverse data owners and users. Although simple, the traditional encryption schemes involve complex key management protocols and, hence, are not apt for data sharing. Proxy re-encryption (PRE), a notion first proposed by Blaze et al., allows a proxy to transform a file computed under a delegator's public key into an encryption intended for a delegate. Let the data owner be the delegator and the data user be the delegate. In such a scheme, the data owner can send encrypted messages to the user temporarily without revealing his secret key. The data owner or a trusted third party generates the re-encryption key. A proxy runs the re-encryption algorithm with the key and revamps the ciphertext before sending the new ciphertext to the user. An



intrinsic trait of a PRE scheme is that the proxy is not fully trusted (it has no idea of the data owner's secret key). This is seen as a prime candidate for delegating access to encrypted data in a secured manner, which is a crucial component in any data-sharing scenario. In addition, PRE allows for encrypted data in the cloud to be shared to authorized users while maintaining its confidentiality from illegitimate parties. Data disclosures can be minimized through the use of encryption since only users delegated by the data owner can effectively access the outsourced data. Motivated by this scenario, this article proposes an improvement in IoT data sharing by combining PRE with identity-based encryption (IBE), information-centric networking (ICN), and blockchain technology. Shamir first presented the notion of IBE, in which a sender encrypts a message to a recipient using the identity (email ) as the public key. It is a very powerful primitive used to combat numerous key distribution problems and has consented to the development of several cryptographic protocols, including public-key searchable encryption, secret handshakes, and chosen ciphertext attack (CCA) secure public-key encryption. IBE is preferred over attribute-based encryption (ABE) because ABE involves heavy computations on data encryption, decryption, and key management, and these processes are not convenient for the resource-constrained IoT devices. The strength of this article is increased by borrowing the idea of ICN to cater for the growth in information sharing. The appeal for low-latency applications introduced the notion of ICN, where data owners can distribute and assign unique names to their data which can be replicated and saved in network caches. This ensures that there is an efficient data delivery and utilization of network bandwidth, which is a prerequisite for the IoT ecosystem regardless of the enormous growth in network volumes. On issues of trust, a decentralized, distributed system that can smoothen secure and trusted data sharing was introduced by Nakamoto. This is the blockchain technology, and it has gained much attention due to its ability to preserve data privacy. Although there exist optimization issues when storing vast sizes of data, emerging system applications have used the blockchain for access control in database management. Data confidentiality and user revocation can also be achieved using blockchain.





## 1.2 Objective

The objective we propose a proxy re-encryption approach to secure data sharing in cloud environments. Data owners can outsource their encrypted data to the cloud using identity-based encryption, while proxy re-encryption construction will grant legitimate users access to the data.

## 1.3 Scope of the Project

In this paper, in data sharing, any information must be encrypted from the source and only decrypted by authorized users in order to preserve its protection. The use of symmetric encryption implies that the same key is shared between the data owner and users, or at least the participants agree on a key. Furthermore, the data owners do not know in advance who the intended data users are, and, therefore, the encrypted data needs to be decrypted and subsequently encrypted with a key known to both the data owner and the users

## 1.4 Existing System:

- Cloud storage services can implement CL-PRE to allow users to securely share files with others. This is particularly useful in collaborative environments where data confidentiality is crucial.
- While CL-PRE aims to resolve the key escrow problem, it still requires careful management of private keys and public parameters. If these keys are compromised, it can jeopardize the entire system's security.
- This raises concerns about its resilience against sophisticated attacks, especially as new vulnerabilities are discovered.

### 1.4.1 Existing System Disadvantages:

- This could potentially allow unauthorized users to access sensitive data.
- As the number of users increases, managing keys and parameters can become cumbersome



## CHAPTER 2

### LITERATURE SURVEY

#### 1.1 Literature Survey

**TITLE:** “Hybrid attribute-and re-encryption based key management for secure and scalable mobile applications in clouds.

**AUTHOR:** P. K. Tysowski and M. A. Hasan

**YEAR:** 2023

**DESCRIPTION:**

Outsourcing data to the cloud are beneficial for reasons of economy, scalability, and accessibility, but significant technical challenges remain. Sensitive data stored in the cloud must be protected from being read in the clear by a cloud provider that is honest-but-curious. Additionally, cloud-based data are increasingly being accessed by resource-constrained mobile devices for which the processing and communication cost must be minimized. Novel modifications to attribute-based encryption are proposed to allow authorized users access to cloud data based on the satisfaction of required attributes such that the higher computational load from cryptographic operations is assigned to the cloud provider and the total communication cost is lowered for the mobile user. Furthermore, data re-encryption may be optionally performed by the cloud provider to reduce the expense of user revocation in a mobile user environment while preserving the privacy of user data stored in the cloud. The proposed protocol has been realized on commercially popular mobile and cloud platforms to demonstrate real-world benchmarks that show the efficacy of the scheme. A simulation calibrated with the benchmark results shows the scalability potential of the scheme in the context of a realistic workload in a mobile cloud computing system.



**TITLE:** Decentralizing privacy: Using blockchain to protect personal data.

**AUTHOR:** E. M. Kornaropoulos, N. Moyer, C. Papamanthou, and A. Psomas,

**YEAR:** 2023

**DESCRIPTION:**

The rise in surveillance and security breaches highlights flaws in the current model where third parties control large amounts of personal data. Bitcoin has shown that decentralized, auditable computing is viable through a peer-to-peer network and public ledger. This paper proposes a decentralized personal data management system that gives users ownership and control of their data. We present a protocol that transforms a blockchain into an automated access-control manager, eliminating the need for trusted intermediaries. Unlike Bitcoin, transactions in our system carry data-related instructions—such as storing, querying, and sharing. We also explore future blockchain enhancements for broader trusted computing applications.

**TITLE:** An access control mechanism to ensure privacy in named data networking using attribute-based encryption with immediate revocation of privileges

**AUTHOR:** R. S. Da Silva and S. D. Zorzo

**YEAR:** 2023

**DESCRIPTION:**

For future Internet, information-centric networking (ICN) is considered a potential solution to many of its current problems. However, concern regarding the protection of user data persists. This paper presents an access control mechanism that will allow users to set fine-grained access policies for applications in named data networking (NDN), a popular ICN architecture. Using an attribute-based encryption scheme with an immediate revocation of privileges, data security is guaranteed. The mechanism inserts a Cloud Proxy Server to mediate the access to the protected data and to inspect for revocation. As an optional feature, the NDN router can add Cloud Proxy Server functions. According to the experiments, the proposed security mechanism proved functional in terms of processing time, memory usage, and file size, which influence both storage and transmission and demonstrate efficiency in manipulating dozens of attributes in an access policy.



**TITLE:** A Medical Data Sharing Scheme Based on Blockchain Attribute-Based Searchable Encryption

**AUTHOR:** T. Chen et al.

**YEAR:** 2023

**DESCRIPTION:**

Aiming at the problems of low sharing efficiency and easy privacy leakage of electronic medical records, a blockchain-based attribute-based searchable encrypted electronic medical record sharing scheme is proposed to achieve fine-grained access control of users' search privileges in one-to-many data sharing scenarios, and at the same time, solve the inefficiency of ciphertext retrieval. In this paper, the shared medical data is divided into structured and unstructured data, and the data is stored in a combined on-chain and off-chain mode, which ensures that both participate in the sharing at the same time, and also relieves the storage pressure of the blockchain. Attribute encryption based on ciphertext policy provides fine-grained access control services, and a data dictionary index structure is designed to complete ciphertext search in the index using searchable encryption. Blockchain nodes retrieve ciphertext keywords and search the index to improve the search efficiency of blockchain nodes. Experimental analysis shows that the scheme can quickly perform the search and sharing of electronic medical records.

**TITLE:** “Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography

**AUTHOR:** E. Barker, L. Chen, S. Keller, A. Roginsky, A. Vassilev, and R. Davis

**YEAR:** 2021

**DESCRIPTION:**

This Recommendation specifies key-establishment schemes based on the discrete logarithm problem over finite fields and elliptic curves, including several variations of Diffie-Hellman and Menezes-QuVanstone(MQV) key establishment schemes.

.



**TITLE:** A Revocable Certificateless Sanitizable Signature Scheme With Batch Verification

**AUTHOR:** E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia

**YEAR:** 2022

**DESCRIPTION:**

In medical and healthcare application scenarios, data integrity and user privacy are often concerned the most. A sanitizable signature scheme is a common adopted approach since it admits an authorized person to sanitize partial sensitive information of signed messages while keeping the correctness of preserved message and signature pairs. A certificateless sanitizable signature scheme further enjoys the merits of certificateless public key systems. That is, it is unnecessary to maintain the public key certificates in traditional cryptosystems or deal with the key escrow problem in identity-based systems. However, the functionality of batch verification for retained message-signature pairs and the revocation mechanism are still lack in previous works. To resolve these issues, the authors propose a revocable certificateless sanitizable signature scheme with batch verification in this paper. The revocation mechanism is fulfilled by combining user's private key with an updatable time key. Hence, a revoked user is unable to request the renewed time key. We also provide a formal proof that our proposed system satisfies the security requirement of existential unforgeability under adaptive chosen-message attacks (EUF-CMA) in the random oracle model. In comparison to related variants, our system demonstrates superior functionality and computational efficiency.



## CHAPTER 3

### DESIGN AND DEVELOPMENT

#### 3.1 Problem Statement

Certificate-less Proxy Re-Encryption (CL-PRE) is a cryptographic approach that combines the advantages of public key infrastructure and identity-based encryption while addressing the key escrow problem found in identity-based systems. It eliminates the need for digital certificates and simplifies key management, making it suitable for secure data sharing. However, its complex cryptographic mechanisms can make implementation and understanding more difficult compared to traditional schemes. CL-PRE also relies on a trusted authority to generate and distribute system parameters, which creates a single point of failure. If compromised, this authority could undermine the system's security. Additionally, the security proofs supporting CL-PRE are less mature than those of established models, raising concerns about its resilience against advanced attacks.

#### 3.2 System Architecture

##### 3.2.1 Proposed System

- Data that must be accessed from the cloud must be protected. However, cloud owners and users have a big hurdle in terms of security and personal data privacy.
- Due to the data owners' lack of trust, they save their data in an encrypted format that is inaccessible to outsiders.
- The phrase “proxy re-encryption” (PRE) refers to a popular way of delivering encrypted data stored in the cloud.
- When a data owner wants to share encrypted data with both the data user and the cloud server (proxy), the data owner generates re-encryption data and sends it to the proxy, which can use it to convert the data holder's cipher texts into the user's plaintexts without having to look at the plaintexts.

##### 3.2.2 Proposed System Advantages

- It trusted authorities to generate and distribute a parameters
- It Provides a public platform for data owners to store and share their encrypted data.



### 3.2.3 System Architecture

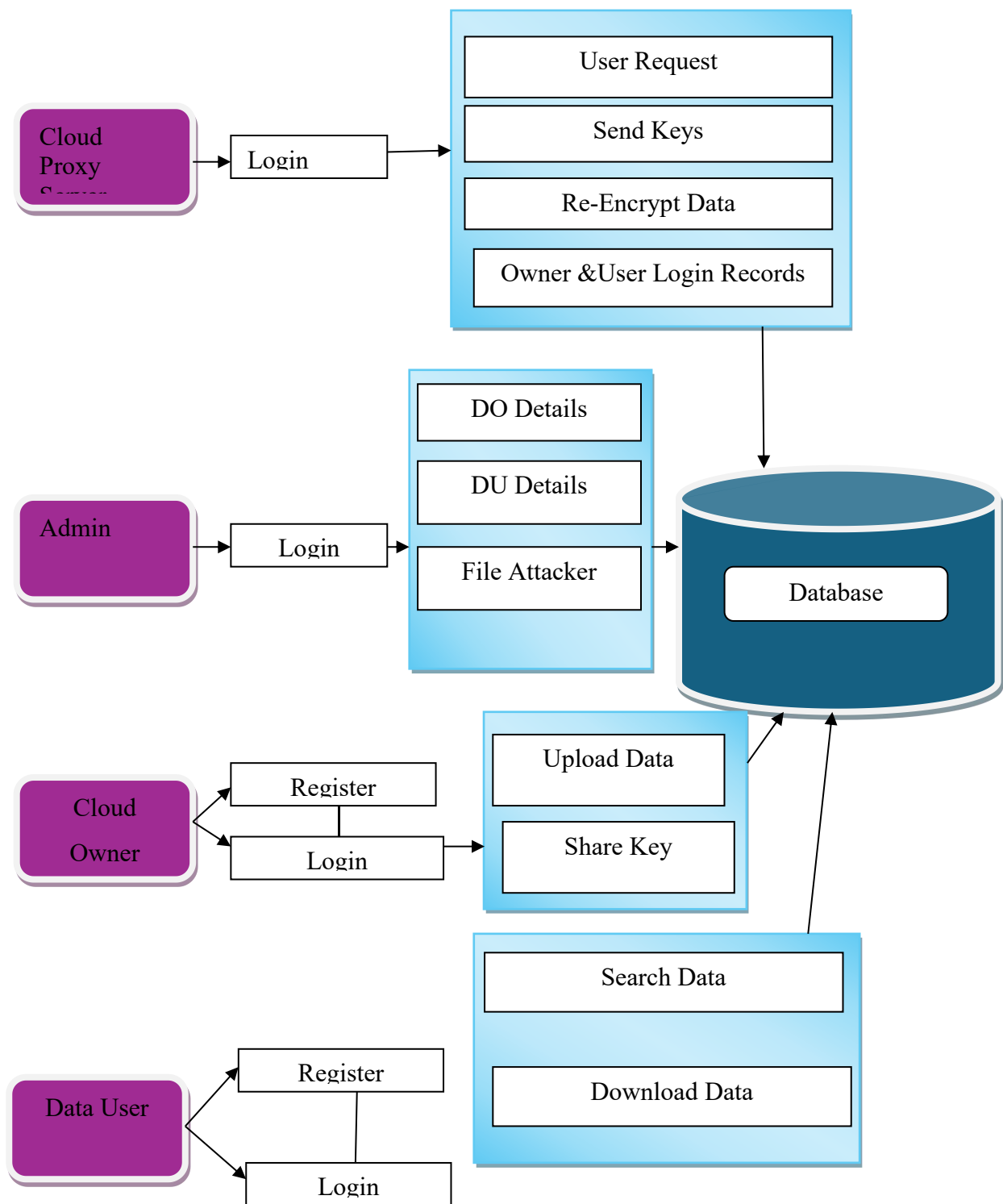


Fig 3.2.3: System Architecture



### 3.3 Design Methodology

#### 3.3.1 Modules

##### 3.3.1.1 User Interface Design

In this module we design the windows for the project. These windows are used for secure login for all users. To connect with server user must give their username and password then only they can able to connect the server. If the user already exists directly can login into the server else user must register their details such as username, password and Email id, into the server. Server will create the account for the entire user to maintain upload and download rate. Name will be set as user id. Logging in is usually used to enter a specific page.

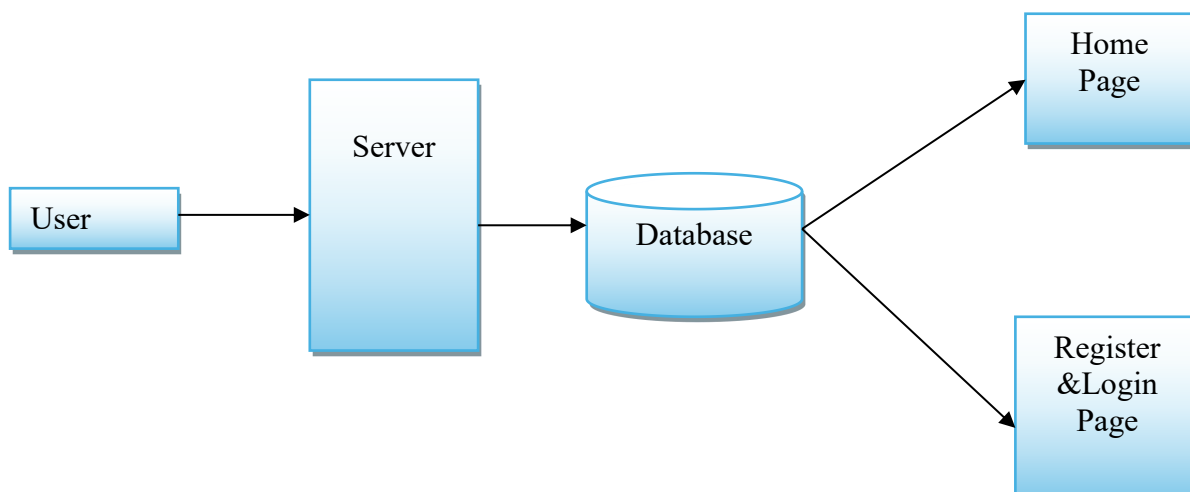


Fig. 3.3.1.1 User Interface Flow Diagram





This is the first module. Here admin has a main module to stores all the information's of the project. Admin is a login with a user id and password. Admin has a stores all information's of details. Admin can also have a data owner detail. Admin can have a data user details in the database. The admin can have a file attacker to file attack information

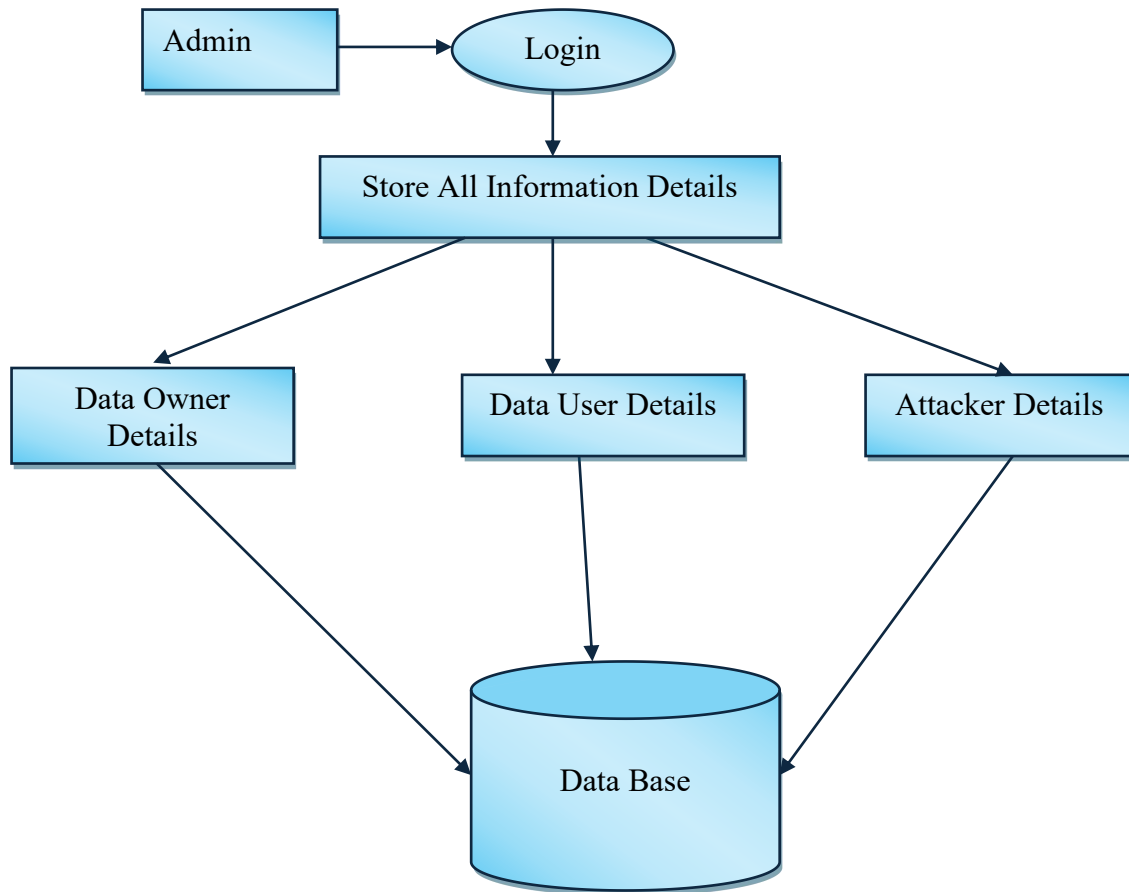


Fig. 3.3.1.2 Admin Interaction with User and Attacker Data

### 3. Cloud Owner



This is the Second module of this project. In this module cloud owner should register and Login. Cloud owner can store a data in a text format. Cloud owner can also have a key send to the Cloud Proxy Server should approve then the key will send an authentic user. User can access a data.

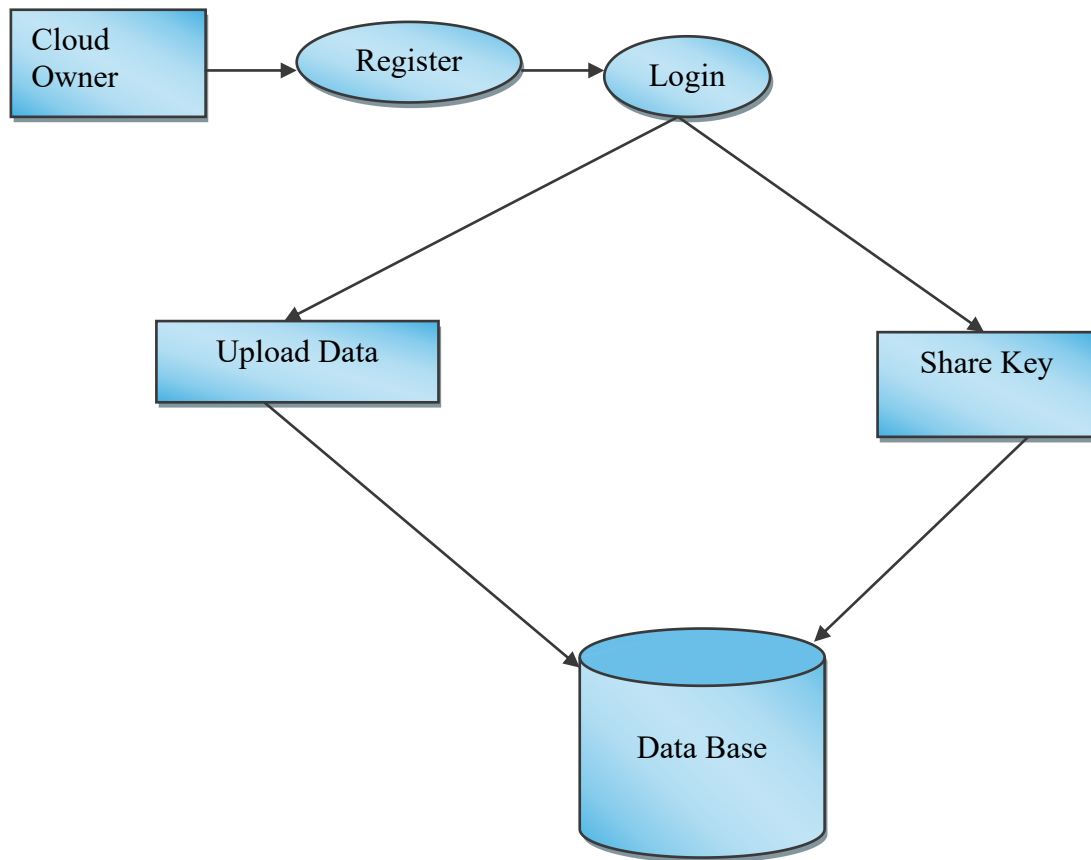


Fig. 3.3.1.3 Cloud Owner Workflow Diagram

#### 4. Cloud Cloud Proxy Server



In this module Cloud Proxy Server has login. After login it will have a data user request. Data user after register it will take a permission from the server. In server it will approve then the data user can login. Cloud Proxy Server has a send key to the user. Cloud Proxy Server can also be re-encrypting a file then forward to the user. Cloud Proxy Server has a key will updates and key will modify. Cloud Proxy Server has a maintains a records of a logins of Cloud Owner and data users.

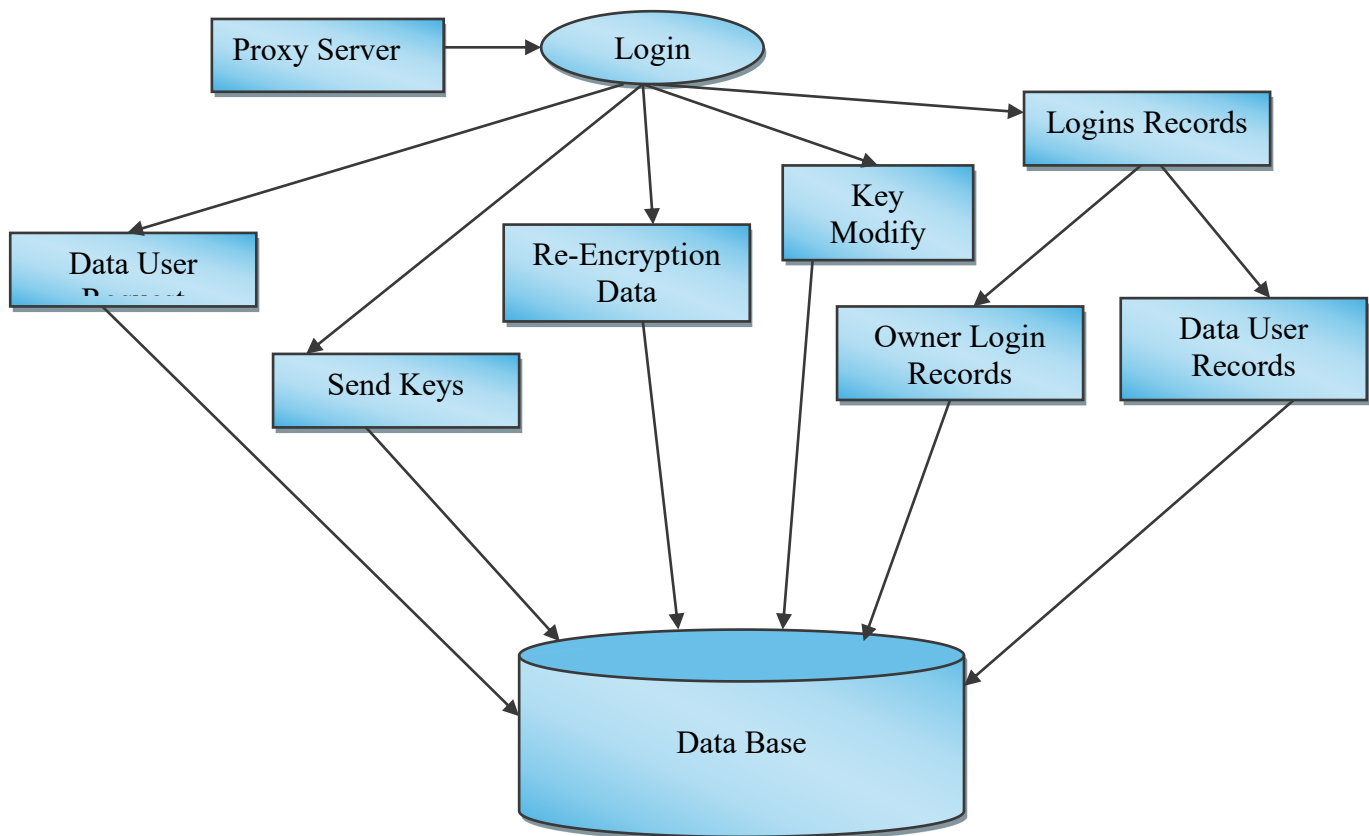


Fig. 3.3.1.4 Data Re-Encryption and Access Control Flow

## 5. Data User



In this module the data user can also a register with the details. Login it takes permission from the Cloud Proxy Server. Cloud Proxy Server can a approve a user. After approve it was data user can login. Data user can also a search a data. Data user can also a download a file.

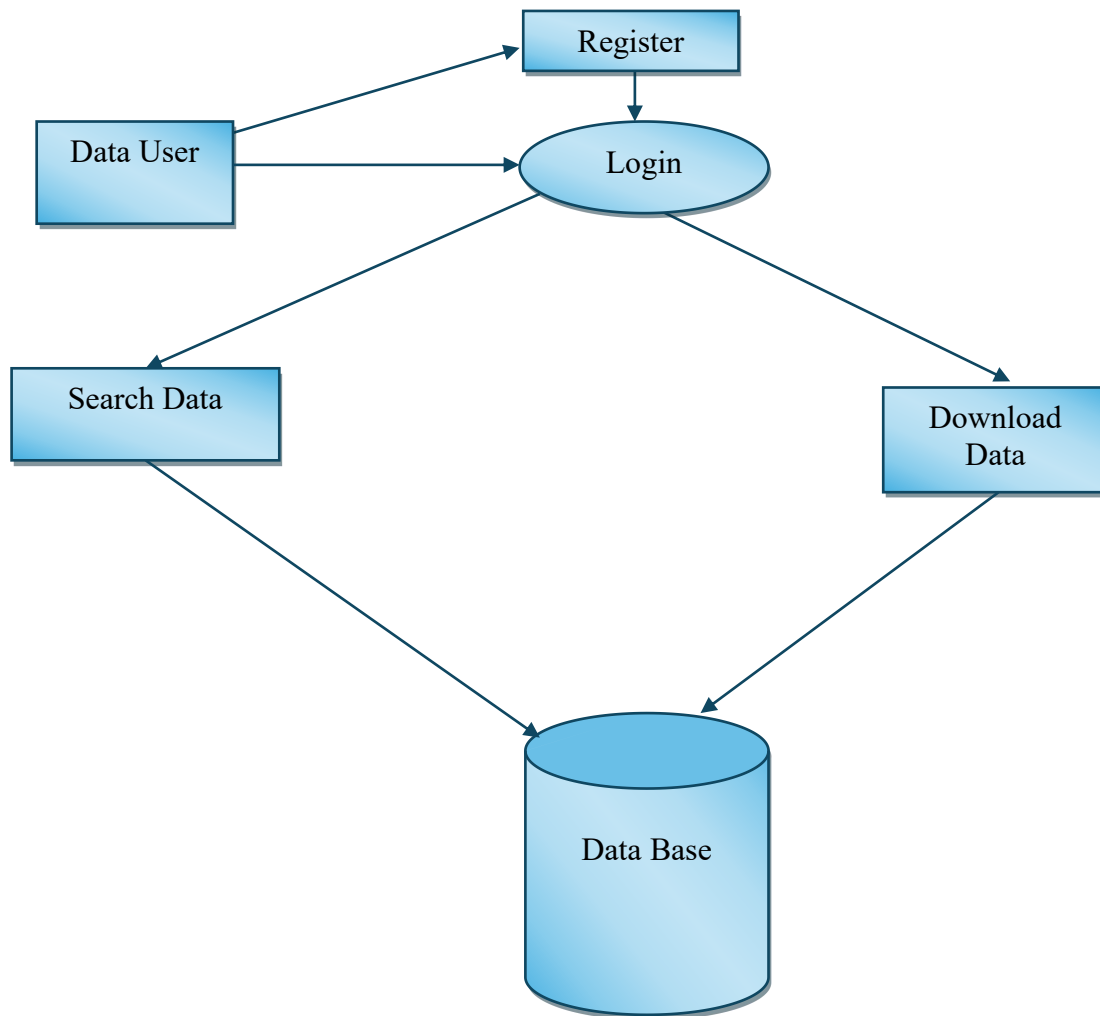


Fig. 3.3.1.5 Data User Functional Flow Diagram

### 3.3.2 Hardware Requirements



The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It should what the system does and not how it should be implemented.

- PROCESSOR : DUAL CORE 2 DUOS.
- RAM : 2GB DD RAM
- HARD DISK : 250 GB

### 3.3.3 Software Requirements

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the teams and tracking the team's progress throughout the development activity.

- FRONT END : (JSP, SERVLET)
- BACK END : MY SQL 5.5
- OPERATING SYSTEM : WINDOWS 10
- IDE : ECLIPSE

### 3.3.4 Functional Requirements

#### 1. Admin Module:

- Admin should be able to log in using a user ID and password.
- Admin can store and manage project information.
- Admin can manage data owner details (add, update, delete).
- Admin can manage data user details (add, update, delete).
- Admin can log file attack information (record details of attacks).

#### 2. Cloud Owner Module:



- Cloud Owner should be able to register and log in.
- Cloud Owner can store data in text format.
- Cloud Owner can generate a key for data access.
- Cloud Owner can send the key to the Cloud Proxy Server for approval.
- Cloud Owner can receive confirmation of key approval from the Cloud Proxy Server.

### **3. Cloud Proxy Server Module:**

- Cloud Proxy Server should have a login mechanism.
- Cloud Proxy Server can manage data user requests.
- Cloud Proxy Server can approve or deny data user registration requests.
- Cloud Proxy Server can send keys to approved data users.
- Cloud Proxy Server can re-encrypt files and forward them to users.
- Cloud Proxy Server can update and modify keys as needed.
- Cloud Proxy Server maintains a log of logins for Cloud Owners and Data Users.

### **4. Data User Module:**

- Data User can register with personal details.
- Data User must obtain permission from the Cloud Proxy Server to log in.
- Data User can log in after receiving approval from the Cloud Proxy Server.
- Data User can search for data stored by the Cloud Owner.
- Data User can download files after proper authentication.

#### **3.3.5 Non-Functional Requirements**



**1. Security:**

- All user credentials must be stored securely (e.g., hashed passwords).
- Data encryption must be implemented for sensitive information.
- Access control mechanisms must be in place to ensure only authorized users can access certain functionalities.

**2. Performance:**

- The system should handle multiple concurrent users without significant performance degradation.
- Response time for user requests (login, data retrieval) should be within acceptable limits

**3. Scalability:**

- The system should be designed to accommodate an increasing number of users and data without requiring significant rework.

**4. Usability:**

- The user interface should be intuitive and easy to navigate for all user roles.
- Documentation should be provided for users to understand how to use the system effectively.

**5. Reliability:**

- The system should be available 99.9% of the time, with minimal downtime for maintenance.
- Data should be backed up regularly to prevent loss.

**6. Maintainability:**

- The codebase should be modular and well-documented to facilitate future updates and maintenance.
- The system should allow for easy integration of new features.

**7. Compliance:**



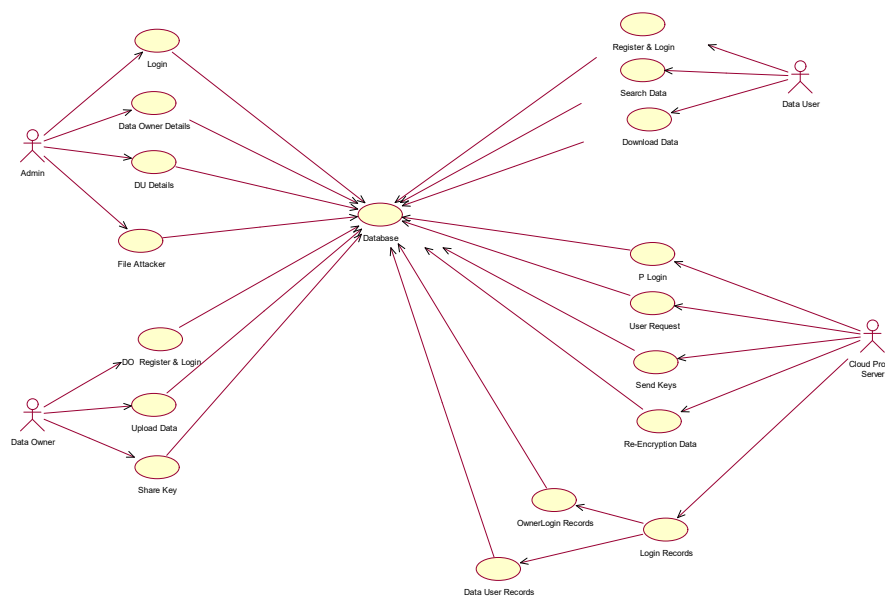
- The system should comply with relevant data protection regulations (e.g., GDPR, HIPAA) depending on the nature of the data being handled.

## 8. Interoperability:

- The system should be able to integrate with other systems or services as needed (e.g., third-party authentication services).

## 3.4 Component Design

### 3.4.1 Use Case Diagram

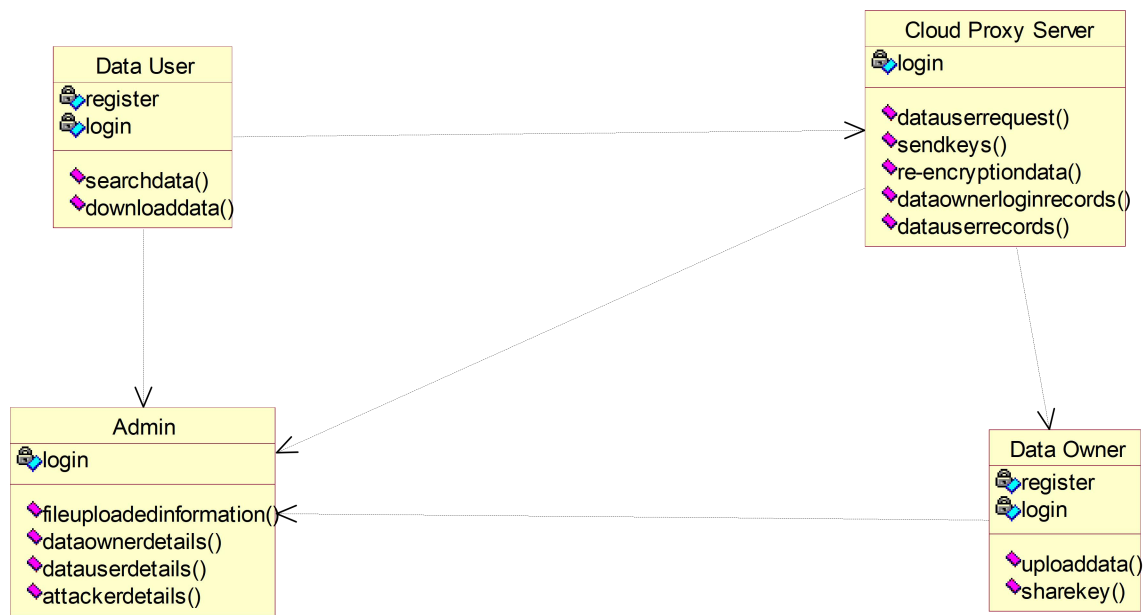


### Explanation:

Place the actors (Admin, Cloud Owner, Cloud Proxy Server, Data User) outside the system boundary, represented by a rectangle. The use cases inside the boundary depict the system's functionalities. The admin can log in, manage data owners and users, and log file attacks. The Cloud Owner can register, log in, store data, generate keys, and send them to the Cloud Proxy Server for approval. The Cloud Proxy Server can log in, approve user requests, send keys, re-encrypt files, and maintain login records. The Data User can register, log in, search data, and download files.

### 3.4.2 Class Diagram



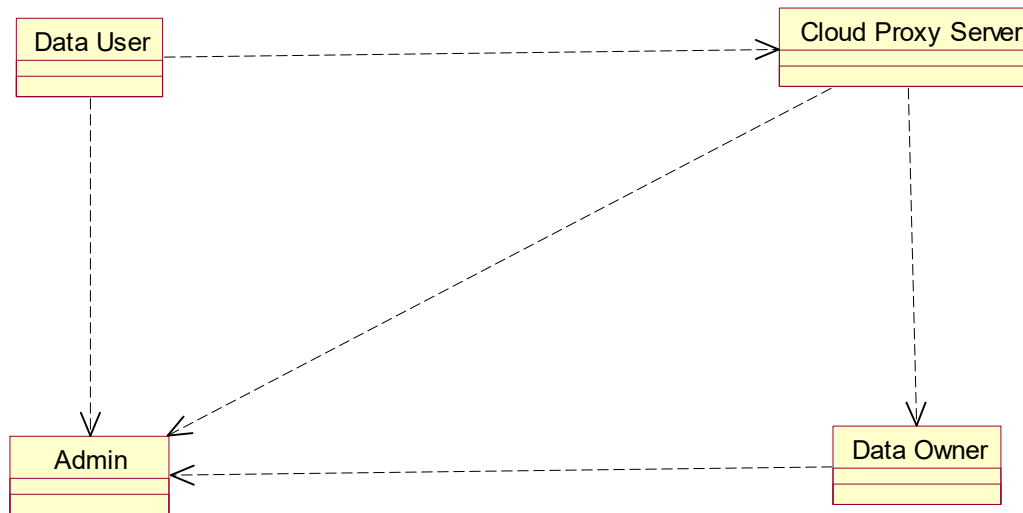


### Explanation:

A class diagram is a fundamental component of object-oriented modelling that provides a static view of a system by illustrating its classes, their attributes, methods, and the relationships between them. It serves as a blueprint for the system's architecture, enabling developers to visualize the structure of the system and understand how different components interact with one another. In the context of a project involving multiple roles such as Admin, Cloud Owner, Cloud Proxy Server, and Data User, a class diagram becomes an essential tool for capturing the system's design and ensuring that all necessary components are well-defined.

At its core, a class diagram consists of several key elements: classes, attributes, methods, and relationships. Classes represent the fundamental building blocks of the system, encapsulating both data and behaviour. Each class is defined by its attributes, which represent the data it holds, and its methods, which define the operations that can be performed on that data. For instance, in our project, we might have classes such as Admin, Cloud Owner, Cloud Proxy Server, and Data User. Each of these classes would encapsulate specific attributes and methods relevant to their respective roles.

### 3.4.3 Object Diagram

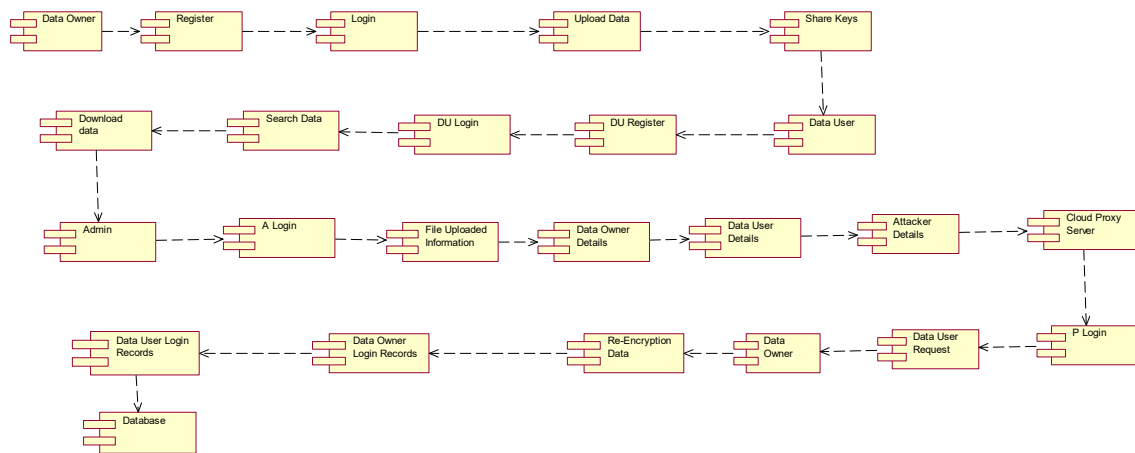


### Explanation:

At its core, an object diagram consists of objects, their attributes, and the relationships between them. Each object is an instance of a class defined in the class diagram, and it represents a specific entity within the system. For example, in a project involving roles such as Admin, Cloud Owner, Cloud Proxy Server, and Data User, an object diagram might include instances of these classes, such as admin1, cloudOwner1, proxyServer1, and dataUser 1. Each of these objects would have specific attribute values that reflect their current state.

For instance, the admin1 object might have attributes like adminID: 001, username: "adminUser ", and password: "securePassword", representing the specific credentials of that Admin instance. Similarly, the cloudOwner1 object could have attributes such as cloudOwnerID: 1001, dataStorage: "dataFile.txt", and encryptionKey: "abc123", indicating the specific data and encryption key associated with that Cloud Owner. By representing these objects and their attributes, the object diagram provides a concrete view of how the system is instantiated at a particular moment.

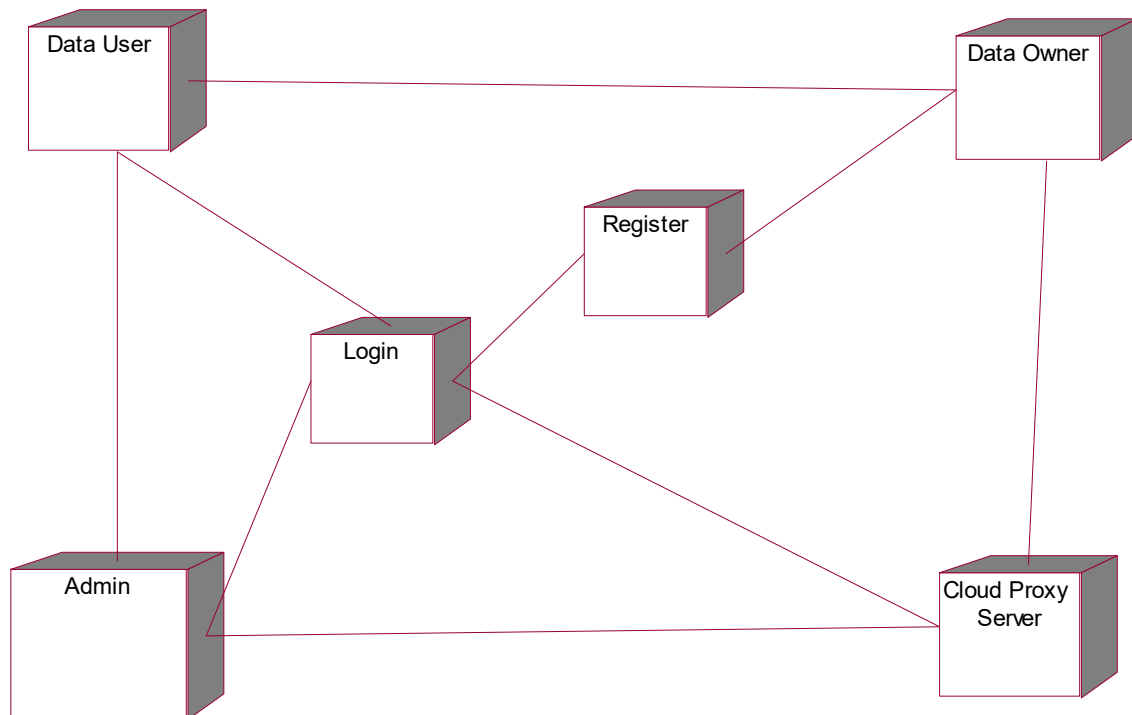
### 3.4.4 Component Diagram



### Explanation:

The User Management component, for example, could be responsible for handling user registration, login, and profile management for both Admins and Data Users. The Data Storage component would manage the storage and retrieval of data, while the Authentication component would handle the security aspects, such as verifying user credentials and managing access tokens. The Cloud Proxy Server component would act as an intermediary, facilitating communication between Data Users and Cloud Owners, while the Data Access component would provide the necessary interfaces for Data Users to search and download files.

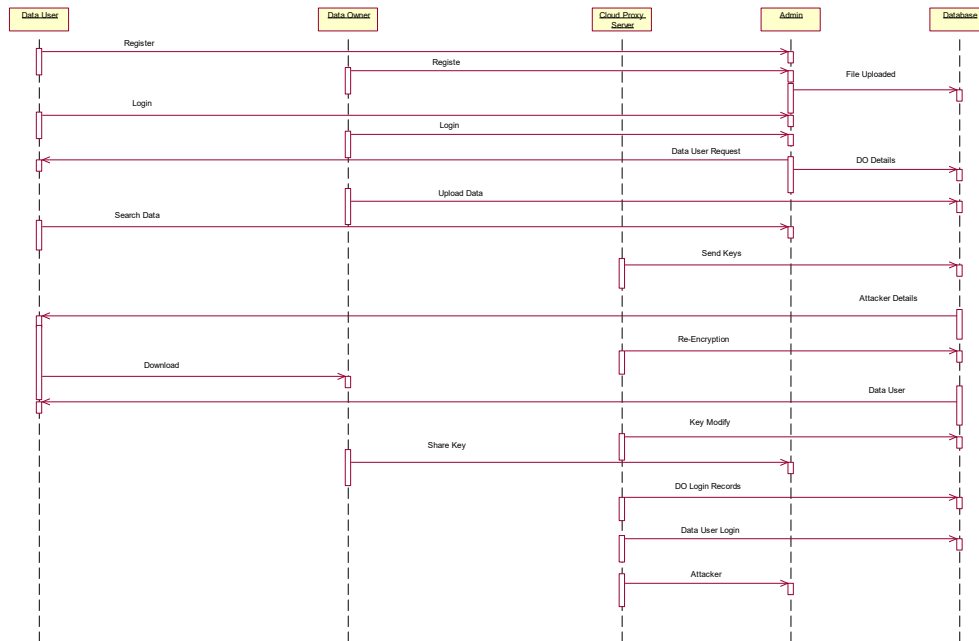
### 3.4.5 Deployment Diagram



### Explanation:

The connections in a deployment diagram illustrate the communication paths between nodes. These connections can represent various types of communication, such as network connections, data flows, or message exchanges. For instance, a connection might exist between the Client Machine and the Web Server, indicating that users access the web application through their browsers. Similarly, there may be connections between the Web Server and the Application Server, as well as between the Application Server and the Database Server, representing the flow of data and requests throughout the system.

### 3.4.6 Sequence Diagram

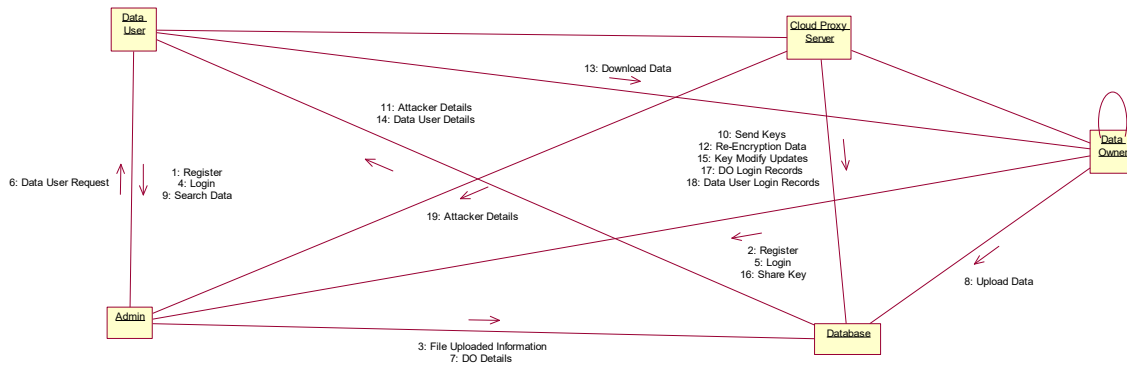


### Explanation:

A sequence diagram is a type of UML (Unified Modeling Language) diagram that illustrates how objects interact in a particular scenario of a use case, emphasizing the order of messages exchanged between them over time. It provides a dynamic view of the system, showcasing the flow of control and data among various components as they collaborate to achieve a specific goal. In projects involving multiple roles such as Admin, Cloud Owner, Cloud Proxy Server, and Data User, a sequence diagram can effectively represent the interactions that occur during specific processes, such as user authentication or data retrieval.

At its core, a sequence diagram consists of lifelines, messages, and activation boxes. Lifelines represent the different objects or components involved in the interaction, depicted as vertical dashed lines. Each lifeline corresponds to a specific instance of a class, such as Admin, Cloud Owner, Cloud Proxy Server, or Data User. The lifelines are arranged horizontally at the top of the diagram, with each object's name displayed above its corresponding line.

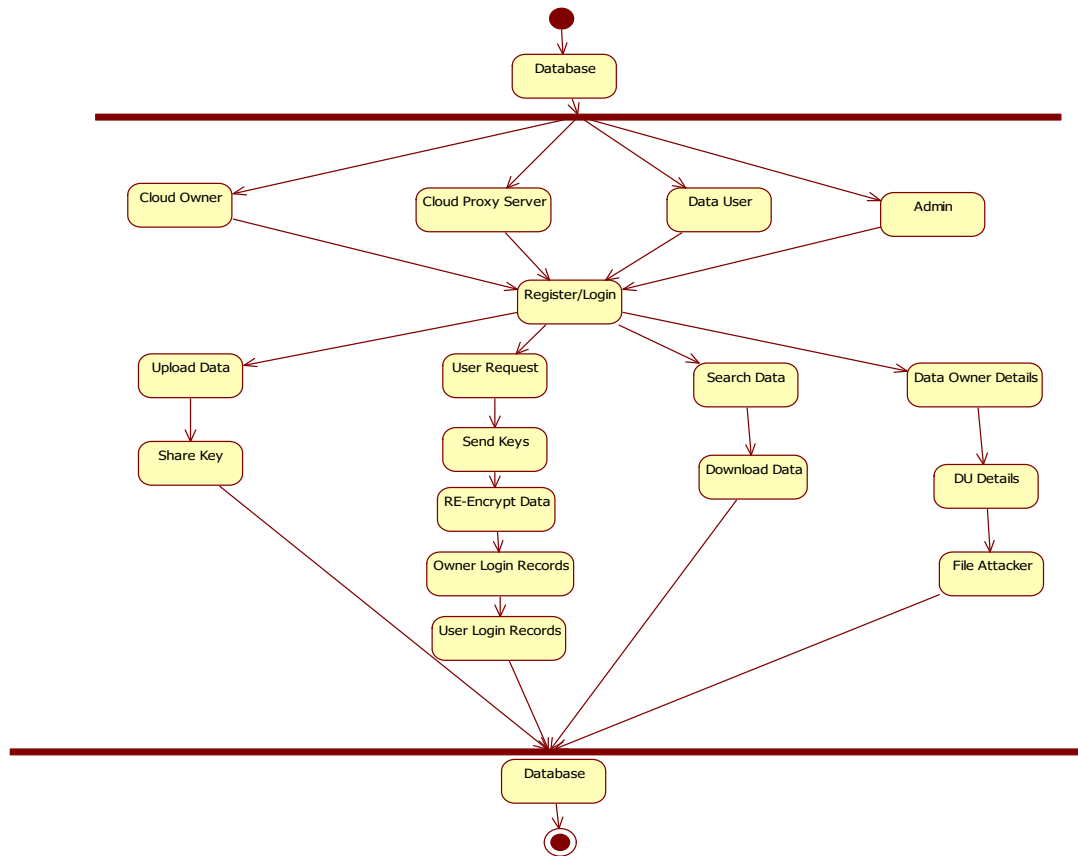
### 3.4.7 Collaboration Diagram



### Explanation:

At its core, a collaboration diagram consists of objects, links, and messages. Objects represent instances of classes that participate in the interaction, depicted as rectangles with the object name and class name. For example, in our project, we might have objects such as admin1: Admin, cloudOwner1: Cloud Owner, proxyServer1: Cloud Proxy Server, and dataUser 1: DataUser . Each object represents a specific instance of a class, and the notation helps clarify the roles of each object in the interaction.

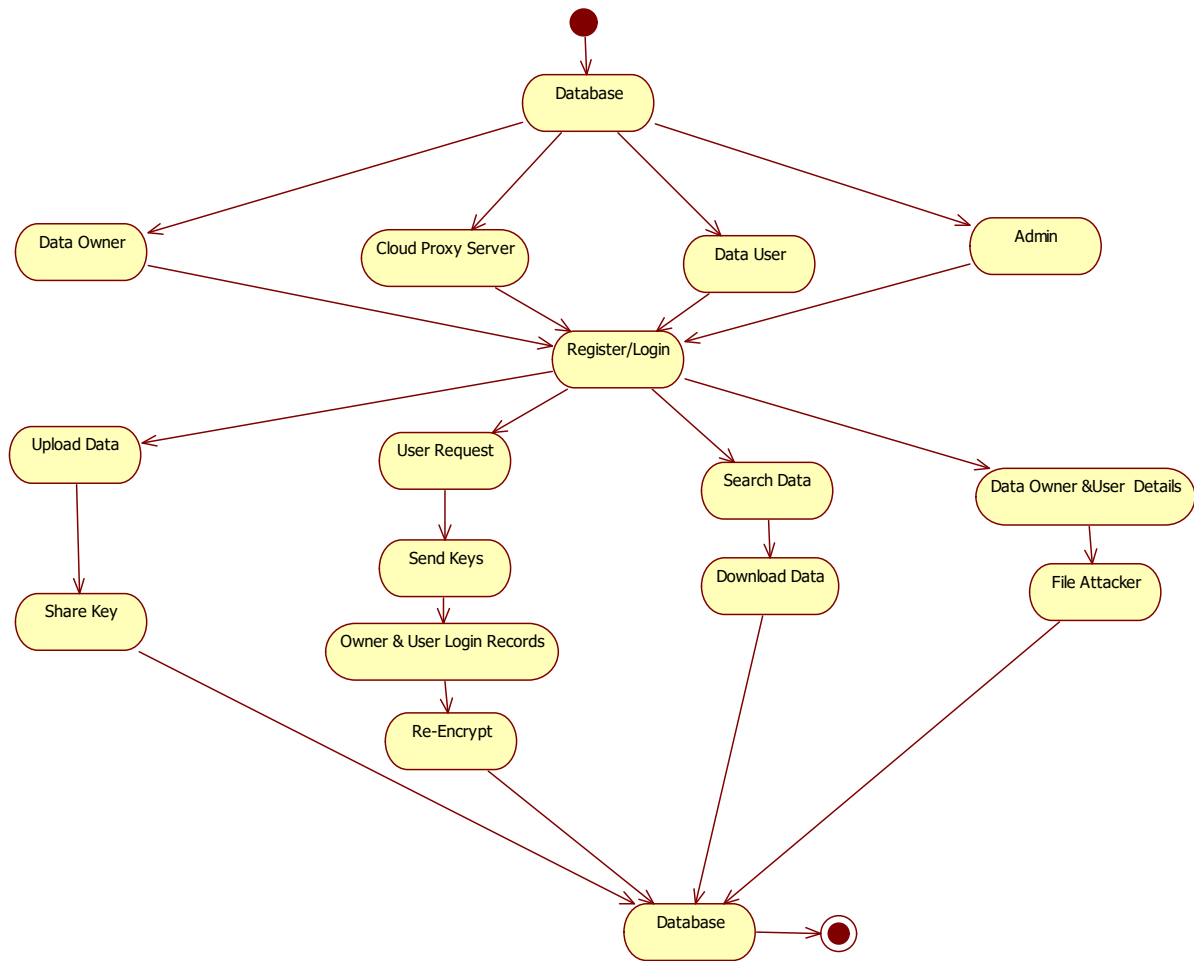
### 3.4.8 State Diagram



### Explanation:

State diagrams are loosely defined diagrams used to illustrate workflows of stepwise activities and actions, including support for choice, iteration, and concurrency. State diagrams require that the system described, a data user, has a login. Once logged in, the data user can perform actions. The data owner also needs to have a login. The data owner has the ability to upload data and view uploaded data. Additionally, the cloud server can perform actions and interact with the database.

### 3.4.9 Activity Diagram



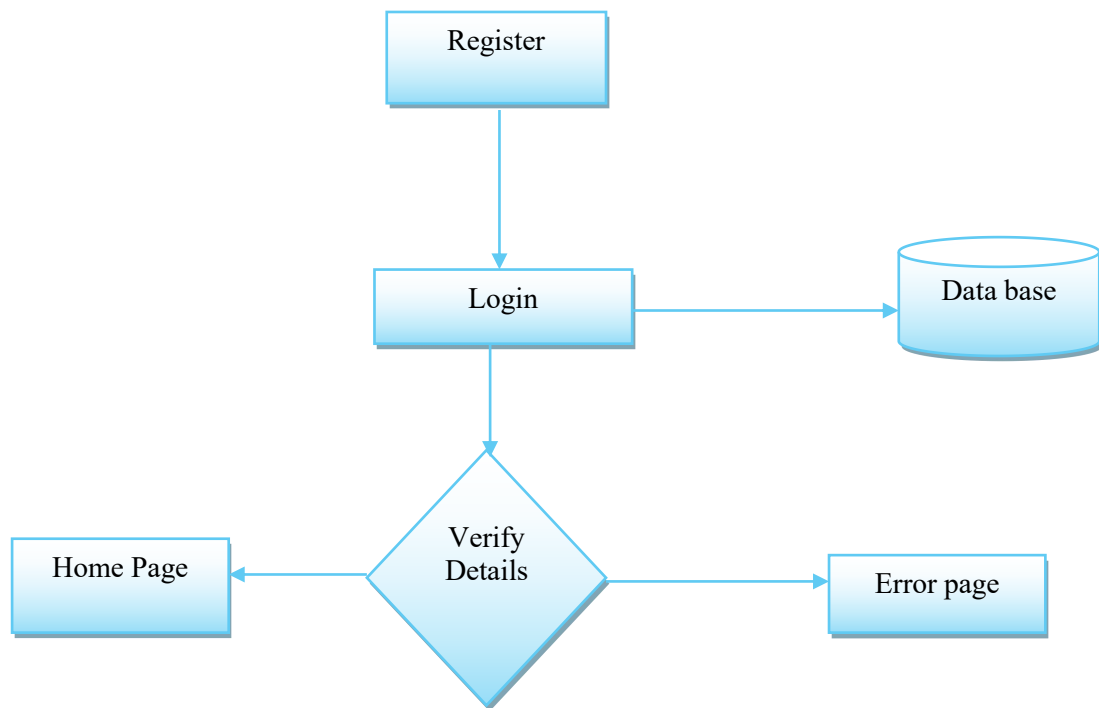
### Explanation:

In projects involving multiple roles such as Admin, Cloud Owner, Cloud Proxy Server, and Data User, a state diagram can effectively represent the lifecycle of key objects, such as user accounts or data access requests.

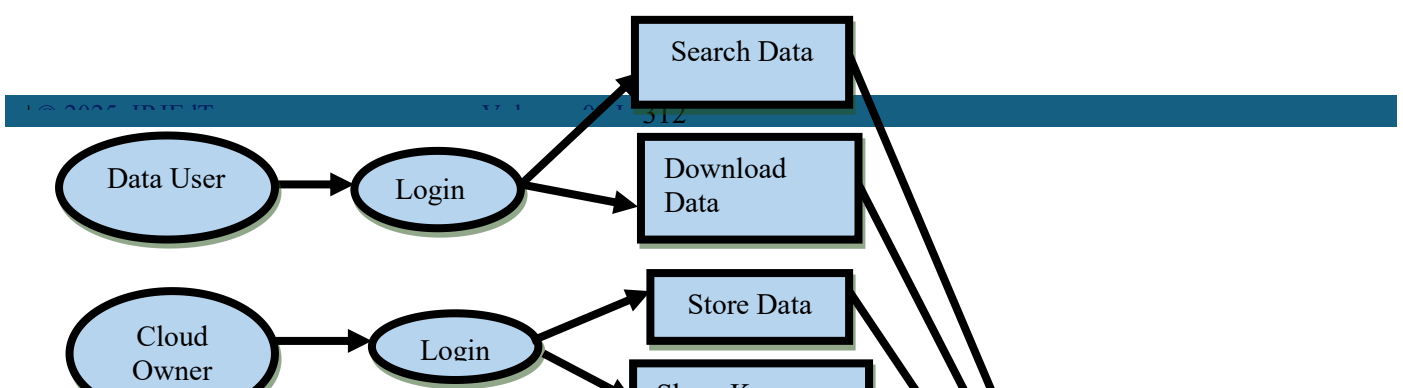
At its core, a state diagram consists of states, transitions, events, and actions. States represent the various conditions or situations that an object can be in at any given time. Each state is depicted as a rounded rectangle, and it can include entry and exit actions that define what happens when the object enters or exits that state. For example, in our project, a `DataUser` might have states such as `Registered`, `Logged In`, `Requesting Data`, and `Downloading Data`. Each of these states reflects a specific condition of the Data User during their interaction with the system.

### 3.4.10 Data Flow Diagram





## Level 1





**Explanation:**



A Data Flow Diagram (DFD) is a graphical representation of the flow of data within a system, illustrating how data is processed by various components and how it moves between them. DFDs are essential tools in systems analysis and design, providing a clear and concise way to visualize the interactions between data sources, processes, and data storage. They help stakeholders understand how information flows through a system, making them invaluable for both technical and non-technical audiences. In projects involving multiple roles such as Admin, Cloud Owner, Cloud Proxy Server, and Data User, a DFD can effectively represent the data interactions and processes that occur within the system.

At its core, a DFD consists of four primary components: processes, data stores, data flows, and external entities.

1. **Processes:** These are the activities or functions that transform input data into output data. Each process is represented by a circle or oval and is typically labelled with a verb-noun phrase that describes the action being performed. For example, in our project, processes might include Authenticate User, Store Data, Retrieve Data, and Log Activity. Each of these processes represents a specific function that the system performs on the data.
2. **Data Stores:** These represent the locations where data is stored within the system. Data stores are depicted as open-ended rectangles or parallel lines and are labelled with descriptive names that indicate the type of data they hold. For instance, a data store might be labeled User Database, Data Files, or Access Logs. Data stores are crucial for understanding where information is kept and how it can be accessed or modified by various processes.
3. **Data Flows:** These are the arrows that indicate the movement of data between processes, data stores, and external entities. Each data flow is labelled with a descriptive name that specifies the type of data being transferred. For example, a data flow might be labelled User Credentials, Data Request, or Access Token. Data flows help visualize how information is exchanged within the system and highlight the relationships between different components.
4. **External Entities:** These represent the sources or destinations of data outside the system. External entities are depicted as rectangles and can include users, other systems, or organizations that interact with the system. For example, in our project, external entities might include Data User, Cloud Owner, and Admin. These entities are crucial for understanding the boundaries of the system and how it interacts with the outside world.

## 3.5 Tools and Technologies Used

### 3.5.1 Proposed Algorithm



## Proxy Re-Encryption Scheme

Proxy re-encryption is a type of public-key encryption that allows a proxy to transform cipher texts from one public key to another, without the proxy having access to the underlying plaintext or private keys. This process enables the proxy to re-encrypt the data without knowing the decryption keys. proxy re-encryption has potential applications for secure sharing in a cloud computing environment. The algorithm is using AES algorithm's we have use a asymmetric algorithm to perform a public and private key to a user. Supports key lengths of 128, 192, or 256 bits, with longer keys providing stronger security. The decryption process is similar but applies the inverse operations in reverse order. Used for encrypting sensitive data in various applications, including file storage and database encryption.

1. **Key Generation:** The data owner generates a public/private key pair.
2. **Encryption:** The data owner encrypts the plaintext using their public key.
3. **Proxy Re-Encryption Key:** The data owner generates a re-encryption key that allows the proxy to transform the ciphertext for a new recipient.
4. **Re-Encryption:** The proxy uses the re-encryption key to convert the original ciphertext into a new ciphertext that can be decrypted by the new recipient's private key.
5. **Decryption:** The new recipient can decrypt the re-encrypted ciphertext using their private key.

## A Simple Proxy Re-Encryption Scheme

Here's a high-level overview of a basic proxy re-encryption scheme:

### Setup

1. **Key Generation:**
  - The data owner generates a public/private key pair:  $((pk, sk))$ .
  - The proxy does not have any keys but can perform re-encryption.
2. **Encryption:**
  - The data owner encrypts a message  $(m)$  using their public key  $(pk)$ :  $[c = \text{Encrypt}(pk, m)]$
  - The ciphertext  $(c)$  is sent to the proxy.
3. **Re-Encryption Key Generation:**



- If the data owner wants to share the data with a new recipient (let's call them Bob), they generate a re-encryption key ( rk ) that allows the proxy to convert the ciphertext for Bob:  $[ rk = \text{GenerateReEncryptionKey}(sk, pk_{\{Bob\}}) ]$
- Here, (  $pk_{\{Bob\}}$  ) is Bob's public key.

#### 4. Re-Encryption:

- The proxy takes the original ciphertext ( c ) and the re-encryption key ( rk ) to produce a new ciphertext ( c' ):  $[ c' = \text{ReEncrypt}(c, rk) ]$
- The new ciphertext ( c' ) can now be sent to Bob.

#### 5. Decryption:

- Bob can decrypt the re-encrypted ciphertext ( c' ) using his private key (  $sk_{\{Bob\}}$  ):  $[ m = \text{Decrypt}(sk_{\{Bob\}}, c') ]$

### Example of a Proxy Re-Encryption Scheme

One of the well-known proxy re-encryption schemes is the **Abe et al. scheme**. Here's a simplified version of how it works:

#### Setup

##### 1. Key Generation:

- The data owner generates a public/private key pair ( (pk, sk) ).
- The public key is used for encryption, and the private key is used for decryption.

##### 2. Encryption:

- The data owner encrypts a message ( m ) using their public key:  $[ c = \text{Enc}(pk, m) ]$

##### 3. Re-Encryption Key Generation:

The data owner generates a re-encryption key ( rk ) for Bob:  $[ rk = \text{GenReKey}(sk, pk_{\{Bob\}}) ]$

##### 4. Re-Encryption:



- The proxy takes the ciphertext (  $c$  ) and the re-encryption key (  $rk$  ) to produce (  $c'$  ):  $[c' = \text{ReEnc}(c, rk)]$

#### 5. Decryption:

- Bob decrypts (  $c'$  ) using his private key:  $[m = \text{Dec}(sk_{\text{Bob}}, c')]$

#### Security Considerations

- **Confidentiality:** The proxy should not learn anything about the plaintext during the re-encryption process.
- **Integrity:** The scheme should ensure that the ciphertext cannot be altered without detection.
- **Non-Transferability:** The re-encryption key should only allow the proxy to re-encrypt for the specified recipient.

#### Applications

- **Data Sharing:** Securely sharing data between users without exposing it to intermediaries.
- **Cloud Storage:** Allowing cloud providers to manage encrypted data without accessing the plaintext.
- **Access Control:** Facilitating dynamic access control in encrypted data systems.

### 3.5.2 Development Tools

#### The Java Framework



Java is a programming language originally developed by James Gosling at Microsystems and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to byte code that can run on any Java Virtual Machine (JVM) regardless of computer architecture. Java is general-purpose, concurrent, class-based, and object-oriented, and is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere".

Java is considered by many as one of the most influential programming languages of the 20th century, and is widely used from application software to web applications the java framework is a new platform independent that simplifies application development internet. Java technology's versatility, efficiency, platform portability, and security make it the ideal technology for network computing. From laptops to datacenters, game consoles to scientific supercomputers, cell phones to the Internet, Java is everywhere!

### **Why Software Developers Choose Java**

Java has been tested, refined, extended, and proven by a dedicated community. And numbering more than 6.5 million developers, it's the largest and most active on the planet. With its versatility, efficiency, and portability, Java has become invaluable to developers by enabling them to:

- Write software on one platform and run it on virtually any other platform
- Create programs to run within a Web browser and Web services
- Develop server-side applications for online forums, stores, polls, HTML forms processing, and more
- Combine applications or services using the Java language to create highly customized applications or services
- Write powerful and efficient applications for mobile phones, remote processors, low-cost consumer products, and practically any other device with a digital heartbeat

### **Object-Oriented**

To be an Object-Oriented language, any language must follow at least the four characteristics.



1. Inheritance: It is the process of creating the new classes and using the behavior of the existing classes by extending them just to reuse the existing code and adding additional feature as needed.
2. Encapsulation: It is the mechanism of combining the information and providing the abstraction.
3. Polymorphism: As the name suggests one name multiple form, Polymorphism is the way of providing the different functionality by the functions having the same name based on the signatures of the methods.
4. Dynamic binding: Sometimes we don't have the knowledge of objects about their specific types while writing our code. It is the way of providing the maximum functionality to a program about the specific type at runtime.

## Java Swing Overview

### Abstract Windows Toolkit (AWT) is Cross-Platform

Swing provides many controls and widgets to build user interfaces with. Swing class names typically begin with a J such as JButton, JList, JFrame. This is mainly to differentiate them from their AWT counterparts and in general is one-to-one replacements. Swing is built on the concept of Lightweight components vs AWT and SWT's concept of Heavyweight components. The difference between the two is that the Lightweight components are rendered (drawn) using purely Java code, such as drawLine and drawImage, whereas Heavyweight components use the native operating system to render the components. Some components in Swing are actually heavyweight components. The top-level classes and any derived from them are heavyweight as they extend the AWT versions. This is needed because at the root of the UI, the parent windows need to be provided by the OS. These top-level classes include JWindow, JFrame, JDialog and JApplet. All Swing components to be rendered to the screen must be able to trace their way to a root window of one of those classes.

**Note:** It generally it is not a good idea to mix heavyweight components with lightweight components (other than as previously mentioned) as you will encounter layering issues, e.g., a lightweight component that should appear "on top" ends up being obscured by a heavyweight component. The few exceptions to this include using heavyweight components as the root pane and for popup windows. Generally speaking, heavyweight components will render on top of lightweight components and will not be consistent with the look and feel being used in Swing. There are exceptions, but that is an advanced topic. The truly





adventurous may want to consider reading this [article](#) from Sun on mixing heavyweight and lightweight components.

### Evolution of Collection Framework:

Almost all collections in Java are derived from the [java.util.Collection](#) interface. Collection defines the basic parts of all collections. The interface states the `add()` and `remove()` methods for adding to and removing from a collection respectively. Also required is the `toArray()` method, which converts the collection into a simple array of all the elements in the collection. Finally, the `contains()` method checks if a specified element is in the collection. The Collection interface is a subinterface of [java.util.Iterable](#), so the `iterator()` method is also provided. All collections have an iterator that goes through all of the elements in the collection. Additionally, Collection is a generic. Any collection can be written to store any class. For example, `Collection<String>` can hold strings, and the elements from the collection can be used as strings without any casting required.

There are three main types of collections:

- Lists: always ordered, may contain duplicates and can be handled the same way as usual arrays
- Sets: cannot contain duplicates and provide random access to their elements
- Maps: connect unique keys with values, provide random access to its keys and may host duplicate values

### List:

Lists are implemented in the JCF via the `java.util.List` interface. It defines a list as essentially a more flexible version of an array. Elements have a specific order, and duplicate elements are allowed. Elements can be placed in a specific position. They can also be searched for within the list. Two concrete classes implement List. The first is `java.util.ArrayList`, which implements the list as an array. Whenever functions specific to a list are required, the class moves the elements around within the array in order to do it. The other implementation is `java.util.LinkedList`. This class stores the elements in nodes that each have a pointer to the previous and next nodes in the list. The list can be traversed by following the pointers, and elements can be added or removed simply by changing the pointers around to place the node in its proper place.

### Set:



Java's [java.util.Set](#) interface defines the set. A set can't have any duplicate elements in it. Additionally, the set has no set order. As such, elements can't be found by index. Set is implemented by [java.util.HashSet](#), [java.util.LinkedHashSet](#), and [java.util.TreeSet](#). [HashSet](#) uses a hash table. More specifically, it uses a [java.util.HashMap](#) to store the hashes and elements and to prevent duplicates. [Java.util.LinkedHashSet](#) extends this by creating a doubly linked list that links all of the elements by their insertion order. This ensures that the iteration order over the set is predictable. [java.util.TreeSet](#) uses a red-black tree implemented by a [java.util.TreeMap](#). The red-black tree makes sure that there are no duplicates. Additionally, it allows Tree Set to implement [java.util.SortedSet](#).

The [java.util.Set](#) interface is extended by the [java.util.SortedSet](#) interface. Unlike a regular set, the elements in a sorted set are sorted, either by the element's `compareTo()` method, or a method provided to the constructor of the sorted set. The first and last elements of the sorted set can be retrieved, and subsets can be created via minimum and maximum values, as well as beginning or ending at the beginning or ending of the sorted set. The [SortedSet](#) interface is implemented by [java.util.TreeSet](#)

[java.util.SortedSet](#) is extended further via the [java.util.NavigableSet](#) interface. It's similar to [SortedSet](#), but there are a few additional methods. The `floor()`, `ceiling()`, `lower()`, and `higher()` methods find an element in the set that's close to the parameter. Additionally, a descending iterator over the items in the set is provided. As with [SortedSet](#), [java.util.TreeSet](#) implements [NavigableSet](#).

### Map:

Maps are defined by the [java.util.Map](#) interface in Java. Maps are simple data structures that associate a key with a value. The element is the value. This lets the map be very flexible. If the key is the hash code of the element, the map is essentially a set. If it's just an increasing number, it becomes a list. Maps are implemented by [java.util.HashMap](#), [java.util.LinkedHashMap](#), and [java.util.TreeMap](#). [HashMap](#) uses a hash table. The hashes of the keys are used to find the values in various buckets. [LinkedHashMap](#) extends this by creating a doubly linked list between the elements. This allows the elements to be accessed in the order in which they were inserted into the map. [TreeMap](#), in contrast to [HashMap](#) and [LinkedHashMap](#), uses a red-black tree. The keys are used as the values for the nodes in the tree, and the nodes point to the values in the map

### Thread:



Simply put, a thread is a program's path of execution. Most programs written today run as a single thread, causing problems when multiple events or actions need to occur at the same time. Let's say, for example, a program is not capable of drawing pictures while reading keystrokes. The program must give its full attention to the keyboard input lacking the ability to handle more than one event at a time. The ideal solution to this problem is the seamless execution of two or more sections of a program at the same time.

### **Creating Threads:**

Java's creators have graciously designed two ways of creating threads: implementing an interface and extending a class. Extending a class is the way Java inherits methods and variables from a parent class. In this case, one can only extend or inherit from a single parent class. This limitation within Java can be overcome by implementing interfaces, which is the most common way to create threads. (Note that the act of inheriting merely allows the class to be run as a thread. It is up to the class to start() execution, etc.)

Interfaces provide a way for programmers to lay the groundwork of a class. They are used to design the requirements for a set of classes to implement. The interface sets everything up, and the class or classes that implement the interface do all the work. The different set of classes that implement the interface have to follow the same rules.

### **3.6 Implementation**

**ProxyReencryption.java**

```

package Algorithm;

import java.io.UnsupportedEncodingException;

import java.math.BigInteger;

public class ProxyReencryption {

    public static void main(String[] args) throws UnsupportedEncodingException {

        char ch[]=new char[16];

        String s = new String();

        BigInteger phi1=new BigInteger("31");

        //Alice private key= "pri1" and public key ="pub1"

        String pri1 = "00000000000000007"; // 128 bit key

        String pub1 = "00000000000000011";

        //Encryption with Alice private and public key

        String Cipher=encrypt(pri1, pub1, "Hellow World");

        System.out.println("encrypted text = "+Cipher);

        // CONVERT Alice public key "pub1" from String to BigInteger data type

        BigInteger pk1= new BigInteger(pub1) ;

        //Bob private key= "pri2" and public key ="pub2"

        String pri2 = "00000000000000013"; // 128 bit key

        String pub2 = "00000000000000017";

        // CONVERT Bob public key "pub2" from String to BigInteger data type

        BigInteger pk2= new BigInteger(pub2) ;

        //Create ProxyKey (proxy_key = bobs_public_key
        *(Alice_public_key.modInverse(random_value).mod(ranom_value)

```



```
BigInteger proxy_key = pk2.multiply(pk1.modInverse(phi1)).mod(phi1);
```

```
//The blow code convert proxy_key which value is "1" to "00000000000000001" 128 bit key
```

```
s= String.valueOf(proxy_key);
```

```
byte[] bytes = s.getBytes("US-ASCII");
```

```
byte[] bytes1 =new byte[16];
```

```
for (int a=0;a<16;a++) {
```

```
    if (a==15)
```

```
        bytes1[a]=bytes[0];
```

```
    else
```

```
        bytes1[a]='0';
```

```
    ch[a]= (char) (bytes1[a]); } }
```

```
String Proxy_re_encryption_key = String.copyValueOf(ch);
```

```
System.out.println("the value of Proxy Re Encryption key =" +Proxy_re_encryption_key);
```

```
//Encryption of Cipher with Proxy Re Encryption key
```

```
String Cipher2=encrypt(pri1, Proxy_re_encryption_key, Cipher);
```

```
System.out.println("2nd encrypted text = " +Cipher2);
```

```
//Decryption the 2nd cipher text with bobs public and private key
```

```
System.out.println("Real Text = " +decrypt(pri2, pub2,Cipher2));
```

```
}
```

```
private static String decrypt(String pri2, String pub2, String cipher2) {
```

```
    // TODO Auto-generated method stub
```

```
    return null;
```

```
}
```



```
private static String encrypt(String pri1, String pub1, String string) {

    // TODO Auto-generated method stub

    return null;

}}
```

### ReEncrypt1.java

```
package com.servlets;

import java.io.IOException;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.DBConnect.ProxyRencrpt.DbConnection;

/**
 * Servlet implementation class ReEncrypt1
 */
@WebServlet("/ReEncrypt1")
public class ReEncrypt1 extends HttpServlet {

    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public ReEncrypt1() {

        super();

        // TODO Auto-generated constructor stub
    }
}
```



}

```
/** * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response) */
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException {
```

```
    String fileId= request.getParameter("fileId");
```

```
    DbConnection db=new DbConnection();
```

```
    try {
```

```
        int i= DbConnection.getRe(fileId);
```

```
        if(i==1){
```

```
            System.out.println("updated ");
```

```
            response.sendRedirect("reencyption.jsp");
```

```
        }else{
```

```
            response.sendRedirect("reencyption.jsp");
```

```
        }
```

```
    } catch (SQLException e) {
```

```
        // TODO Auto-generated catch block
```

```
        e.printStackTrace();
```

```
    }}
```

```
/**
```

```
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
```

```
 */
```

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException {
```

```
    // TODO Auto-generated method stub
```

```
}}
```

### 3.7 Testing and Validation

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of



components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement

### Developing Methodologies

The test process is initiated by developing a comprehensive plan to test the general functionality and special features on a variety of platform combinations. Strict quality control procedures are used. The process verifies that the application meets the requirements specified in the system requirements document and is bug free. The following are the considerations used to develop the framework from developing the testing methodologies.

### Types of Tests

Sl. No	Test scenario	User action	Expected result	Actual Result	Remarks
1.	Registration	Users registering into the system.	Register into the system.	Successfully alert registered message.	Pass
2.	Login	1. Entered correct password.	1. Log into the system. 2. Alert generated.	1. Successfully logged in. 2. Successfully generated the alert.	Pass
3.	Cloud Services	It has an all-details stores in cloud	Messages sending data user alert is generated.	Successfully generated the alert and messages sending	Successful





4.	Cloud Proxy Server	It will have user request and send keys	key has to actions	Successfully generated the alert to Trapdoor key message	Successful
5.	Cloud Owner	Owner Data has a store a file	Messages Alert is generated	Successfully generated the alert for data controller message has displays	Successful
6.	Data User	Data user has a search file and it will have a key response	Messages Alert is generated	Successfully generated the alert for client message has displays	Successful

### 3.7.1 Unit Testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

### 3.7.2 Functional Test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:



- Valid Input : identified classes of valid input must be accepted.
- Invalid Input : identified classes of invalid input must be rejected.
- Functions : identified functions must be exercised.
- Output : identified classes of application outputs must be exercised.
- Systems/Procedures : interfacing systems or procedures must be invoked.

### **3.7.3 System Test**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

### **3.7.4 Performance Test**

The Performance test ensures that the output be produced within the time limits, and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

### **3.7.5 Integration Testing**

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

### **3.7.6 Acceptance Testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.



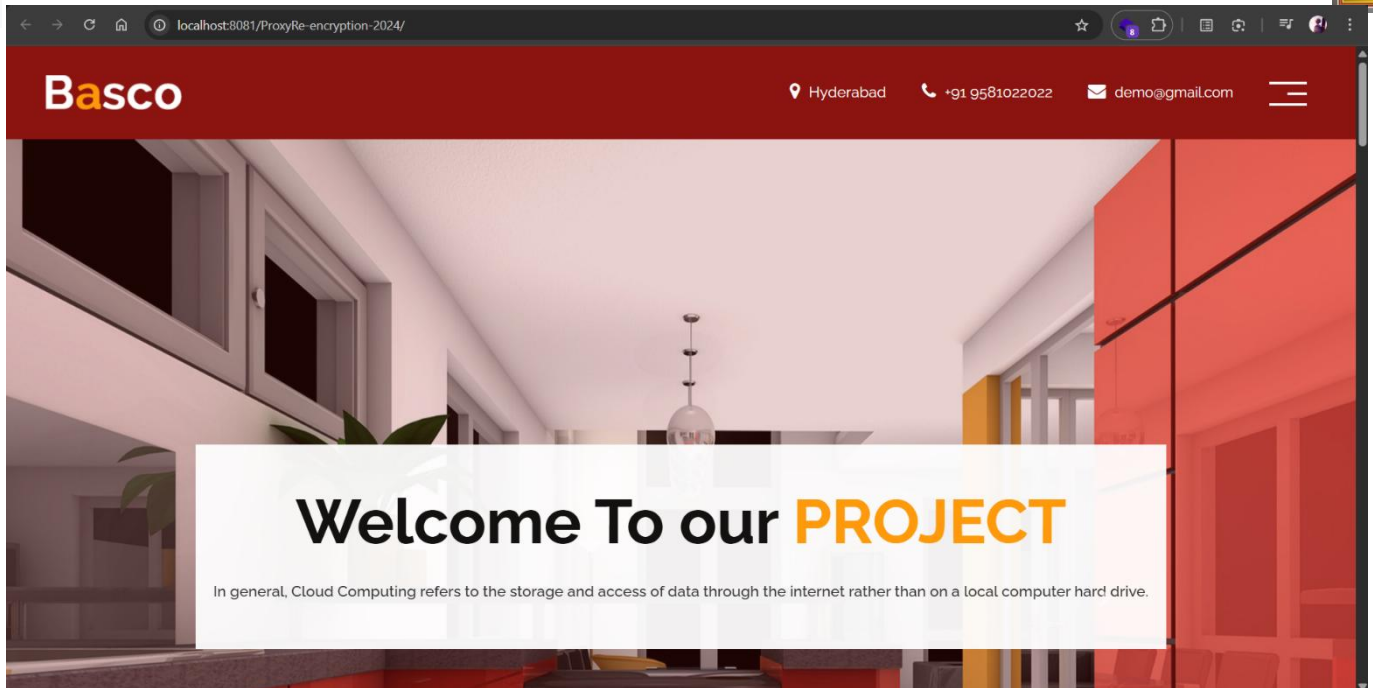
#### **Acceptance Testing for Data Synchronization:**

- The Acknowledgements will be received by the Sender Node after the Packets are received by the Destination Node
- The Route add operation is done only when there is a Route request in need
- The Status of Nodes information is done automatically in the Cache Updation process

#### **3.7.7 Build the Test Plan**

Any project can be divided into units that can be further performed for detailed processing. Then a testing strategy for each of this unit is carried out. Unit testing helps to identify the possible bugs in the individual component, so the component that has bugs can be identified and can be rectified from errors.

### **3.8 Snapshots**



3.8.1 Web Application Interface

3.8.2 Cloud Owner Registration and Login Page



3.8.3 File Upload Page

3.8.4 User Registration Page



3.8.5 User Login Page

3.8.6 File Search Page

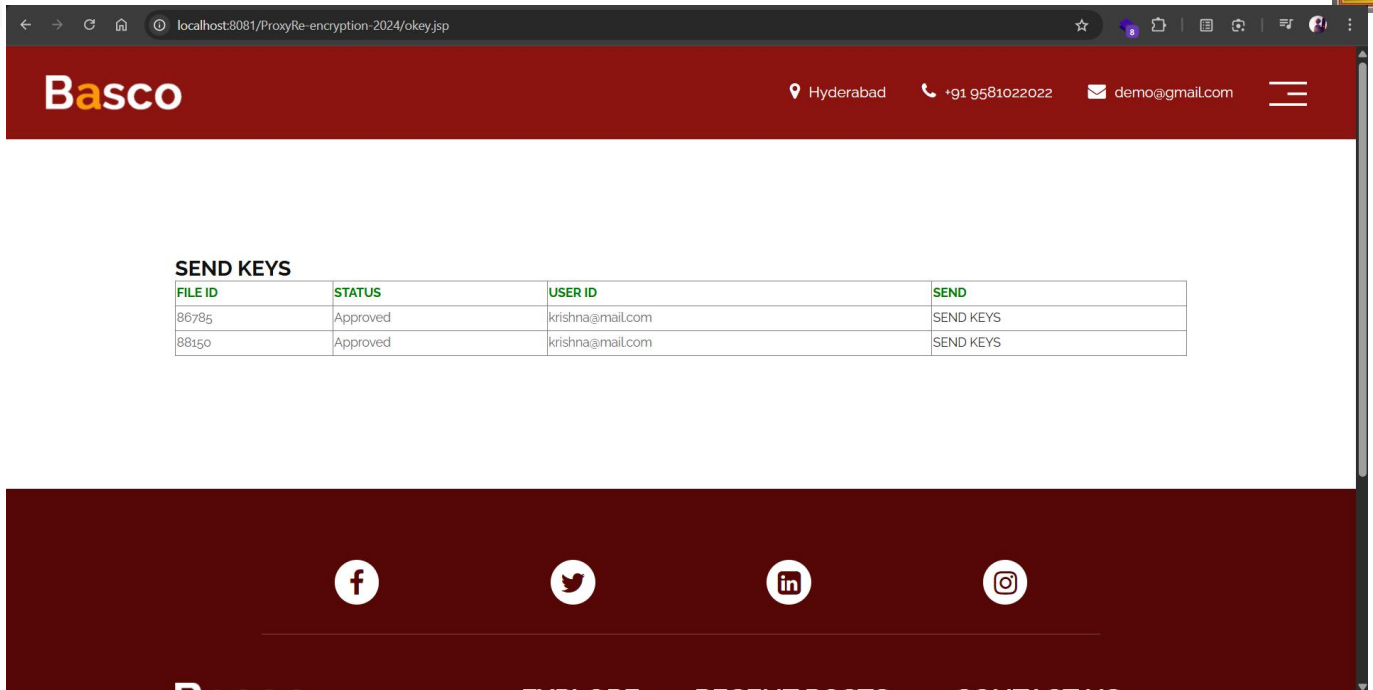


FILE ID	USER ID	FILE NAME	ENCRYPTION	RE- ENCRYPT	KEY REQUEST
88150	abhinav@mail.com	Test file1	Encrypt Data	RE-Encrypt Data	Request

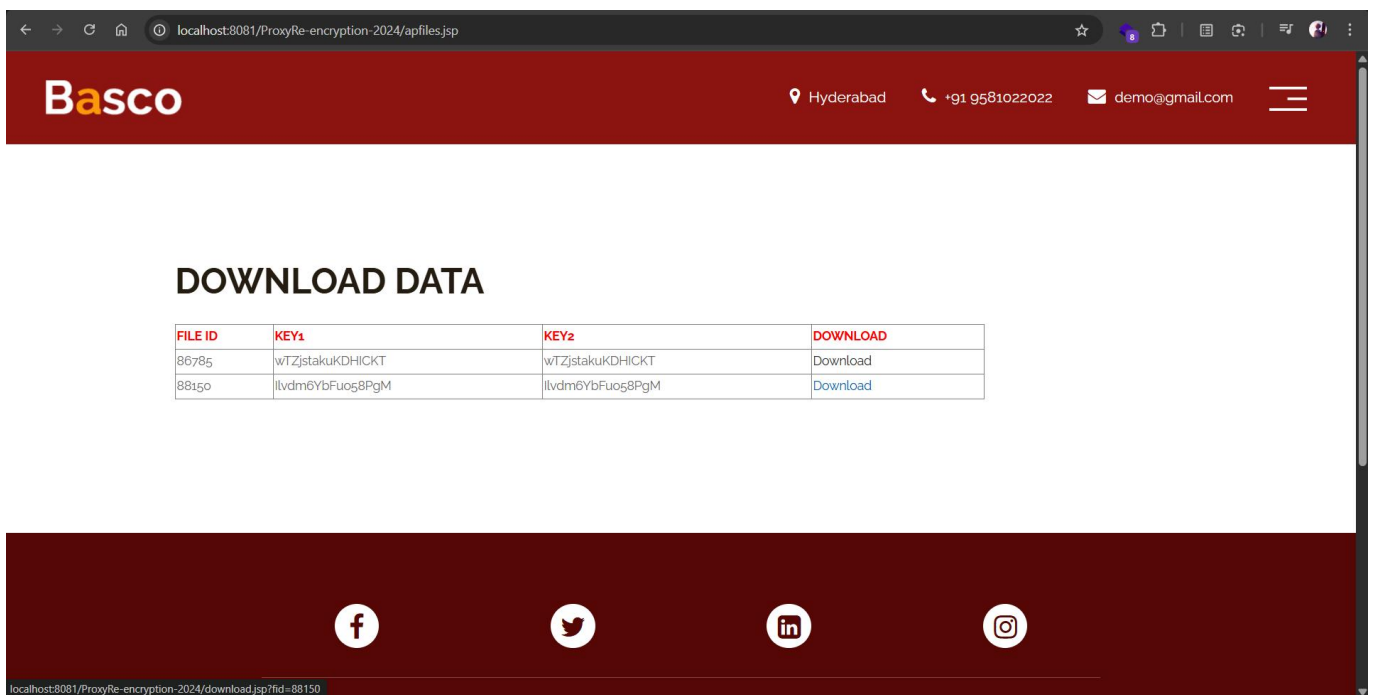
3.8.7 File Request Page

FILE ID	OWNER ID	USER ID	SEND
88150	abhinav@mail.com	krishna@mail.com	APPROVE

3.8.8 Cloud Owner Key Sharing Page



3.8.9 Cloud Proxy Server Key Sending Page



3.8.10 File Download Page for User





## CHAPTER 4

### RESULTS AND DISCUSSION

#### 1.1 Results

The implementation of the proposed system yielded encouraging outcomes, especially in the area of secure data sharing using Proxy Re-Encryption (PRE). The architecture allowed encrypted data to be shared between users through a proxy without revealing the plaintext to the intermediary. This not only preserved data confidentiality but also simplified access delegation. The system incorporated distinct user roles—Admin, Cloud Owner, Cloud Proxy Server, and Data User—each with specific functions and controlled access levels. This role-based access control contributed significantly to the clarity and security of interactions within the system.

System performance was assessed by executing core functionalities such as data uploads by the Cloud Owner, key generation and distribution, re-encryption by the Proxy Server, and data retrieval by the Data User. These tasks were performed without noticeable delays, even under repeated operations, confirming the responsiveness and efficiency of the backend processes. The data remained encrypted throughout transit and storage, reinforcing the system's commitment to privacy and security.

During testing, several test cases were conducted to simulate both valid and invalid user behaviors. The system successfully triggered alerts when unauthorized access attempts were made, validating its basic intrusion detection capability. These attempts were recorded and made available to the Admin, who could take appropriate actions such as reviewing login histories or blocking access. This proactive detection and logging mechanism is essential for cloud-based systems where real-time security monitoring is critical.

Moreover, encryption and key-sharing mechanisms worked as intended. The Cloud Owner generated the keys and passed them through the Proxy Server, which in turn re-encrypted the data for the Data User. At no point did the Proxy Server gain access to the plaintext, preserving end-to-end confidentiality. This confirms that the implemented PRE protocol was both functional and secure.

In conclusion, the results demonstrate that the system meets its objectives effectively. It provides a reliable and secure environment for storing and sharing sensitive data in a cloud infrastructure. The use of



PRE, combined with role-based access and secure key management, ensures that data confidentiality and integrity are maintained while enabling flexible, controlled access for legitimate users.

## 1.2 Future Enhancements

**Blockchain-Based Authorization:** The system can be extended to fully integrate blockchain for decentralized and tamper-proof authorization. This would eliminate the need for a central authority and further improve trust in access control mechanisms.

**Fine-Grained Access Control:** Future versions of the system could enhance attribute-based or role-based access models, allowing even more precise control over who can access which part of the encrypted data.

**Improved Privacy Preservation:** By implementing advanced privacy-preserving techniques such as zero-knowledge proofs or homomorphic encryption, data owners can share information while ensuring that no unnecessary data is exposed.

**Performance Optimization:** The system's efficiency, especially during key re-encryption and user authentication, can be optimized further to reduce latency and improve scalability for large datasets or multiple concurrent users.

**Comparative Evaluation:** More extensive benchmarking and real-time deployment scenarios can be used to evaluate and compare the system's performance against other state-of-the-art data-sharing frameworks, solidifying its practical viability.



## CHAPTER 5

### CONCLUSIONS

#### 5.1 Conclusion

The rapid evolution of the Internet of Things (IoT) has significantly transformed the digital landscape, enabling smart environments and applications that rely heavily on data collection, processing, and sharing. Among these, secure data sharing has become a vital requirement, especially as data flows from devices to cloud infrastructures for storage and access. However, this advancement also introduces critical concerns regarding data confidentiality, integrity, and user privacy. Addressing these concerns requires a robust and flexible data-sharing framework that maintains both security and efficiency.

In this project, we proposed and implemented a secure data-sharing system based on Identity-Based Proxy Re-Encryption (IBPRE) within a cloud computing environment. The core objective was to provide a mechanism that allows data owners to encrypt their data and outsource it to the cloud, while still retaining control over who can access it. With the IBPRE scheme, data owners can delegate access rights to legitimate users without revealing their private keys or directly sharing the original content. This ensures that even the cloud infrastructure, acting as an untrusted intermediary, cannot access the actual data content.

Additionally, the integration of Information-Centric Networking (ICN) principles into the system enhances content delivery by utilizing cached data effectively. This not only reduces data retrieval time and server load but also ensures better utilization of network bandwidth. The ability to deliver content from nearby caches contributes to a more responsive and scalable system, thereby improving the overall quality of service.

Through our implementation and testing, we demonstrated that the proposed scheme not only fulfills its intended security requirements but also performs efficiently under practical conditions. The use of IBPRE, combined with edge computing and ICN, creates a versatile and future-ready framework for secure data sharing in cloud-based IoT environments.

In conclusion, our project successfully showcases a comprehensive solution to the ongoing challenges of secure and efficient data sharing. The framework balances the needs for strong cryptographic protections with the realities of limited device resources and high network demand. This work lays a solid foundation for future advancements in trusted cloud computing and privacy-preserving data access systems.



## CHAPTER 6

### REFERENCES

#### 6.1 References

- [1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Commun. Surveys Tut.*, vol. 17, no. 4, pp. 2347–2376, Oct./Dec. 2015.
- [2] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, Springer, May 1998, pp. 127–144.
- [3] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Proc. Workshop Theory Appl. Cryptographic Techn.*, Springer, Aug. 1984, pp. 47–53.
- [4] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, Springer, May 2004, pp. 506–522.
- [5] B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters, "Building an encrypted and searchable audit log," in *NDSS*, vol. 4. Citeseer, Feb. 2004, pp. 5–6.
- [6] D. Balfanz et al., "Secret handshakes from pairing-based key agreements," in *Proc. IEEE, Symp. Secur. Privacy*, 2003, pp. 180–196.
- [7] R. Canetti, S. Halevi, and J. Katz, "Chosen-ciphertext security from identity-based encryption," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, Springer, 2004, pp. 207–222.
- [8] T. Koponen et al., "A data-oriented (and beyond) network architecture," in *Proc. Conf. Appl., Techn., Architectures, Protec. Compute. Commun.*, Aug. 2007, pp. 181–192.
- [9] N. Fotiou, P. Nikander, D. Trossen, and G. C. Polyzos, "Developing information networking further: From PSIRP to pursuit," in *Proc. Int. Conf. Broadband Commun., Newt. Syst.*, Springer, Oct. 2010, pp. 1–13.
- [10] C. Dannewitz, J. Golic, B. Ohlman, and B. Ahlgren, "Secure naming for a network of information," in *Proc. INFOCOM IEEE Conf. Compute. Commun. Workshops*, 2010, pp. 1–6.
- [11] A. Carzaniga, M. J. Rutherford, and A. L. Wolf, "A routing scheme for content-based networking," in *Proc. IEEE INFOCOM* 2004, vol. 2, 2004, pp. 918–928.
- [12] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *Proc. 2nd ed. ICN Workshop Inform.- Centric Newt.*, Aug. 2012, pp. 55–60.



- [13] Y. Sun et al., “Trace-driven analysis of ICN caching algorithms on video on-demand workloads,” in Proc. 10th ACM Int. Conf. Emerging Newt. Exp. Technol., Dec. 2014, pp. 363–376.
- [14] S. Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, vol. 4. Bitcoin.org, 2008. Available: <https://bitcoin.org/bitcoin.pdf>
- [15] S. Yu, C. Wang, K. Ren, and W. Lou, “Achieving secure, scalable, and fine-grained data access control in cloud computing,” in Proc. IEEE INFOCOM, Mar. 2010, pp. 1–9.
- [16] N. Park, “Secure data access control scheme using type-based reencryption in cloud environment,” in Semantic Methods Knowledge Management and Communications. Berlin, Germany: Springer, 2011, pp. 319–327.
- [17] G. Wang, Q. Liu, J. Wu, and M. Guo, “Hierarchical attribute-based encryption and scalable user revocation for sharing data in cloud servers,” Compute. Secur., vol. 30, no. 5, pp. 320–331, Jul. 2011.
- [18] J. Hur, “Improving security and efficiency in attribute-based data sharing,” IEEE Trans. Knowl. Data Eng., vol. 25, no. 10, pp. 2271–2282, Apr. 2011.
- [19] P. K. Tysowski and M. A. Hasan, “Hybrid attribute-and re-encryptionbased key management for secure and scalable mobile applications in clouds,” IEEE Trans. Cloud Compute., vol. 1, no. 2, pp. 172–186, Nov. 2013.
- [20] Q. Liu, G. Wang, and J. Wu, “Time-based proxy re-encryption scheme for secure data sharing in a cloud environment,” Inform. Sci., vol. 258, pp. 355–370, Feb. 2014.
- [21] J. Han, W. Susilo, and Y. Mu, “Identity-based data storage in cloud computing,” Future Gener. Compute. Syst., vol. 29, no. 3, pp. 673–681, Mar. 2013.
- [22] H.-Y. Lin, J. Kubiawicz, and W.-G. Tzeng, “A secure fine-grained access control mechanism for networked storage systems,” in Proc. IEEE 6th Int. Conf. Soft. Secur. Rel., Jun. 2012, pp. 225–234.
- [23] Y. Zhou et al., “Identity-based proxy re-encryption version 2: Making mobile access easy in cloud,” Future Gener. Compute. Syst., vol. 62, pp. 128–139, Sep. 2016.
- [24] X. A. Wang, J. Ma, F. Xhafa, M. Zhang, and X. Luo, “Cost-effective secure e-health cloud system using identity based cryptographic techniques,” Future Gener. Compute. Syst., vol. 67, pp. 242–254, Feb. 2017.
- [25] J. Shao, G. Wei, Y. Ling, and M. Xie, “Identity-based conditional proxy re-encryption,” in Proc. IEEE Int. Conf. Commun., Jun. 2011, pp. 1–5.
- [26] K. O. B. Obour Agyekum et al., “A secured proxy-based data sharing module in IoT environments using blockchain,” Sensors, vol. 19, no. 5, Jan. 2019, Art. no. 1235.
- [27] G. Zyskind et al., “Decentralizing privacy: Using blockchain to protect personal data,” in Proc. IEEE



Secur. Privacy Workshops, May 2015, pp. 180–184.

[28] D. D. F. Maesa, P. Mori, and L. Ricci, “Blockchain based access control,” in Proc. IFIP Int. Conf. Distributed Appl. Interoperable Syst., Springer, Jun. 2017, pp. 206–220.

[29] K. Fan, Y. Ren, Y. Wang, H. Li, and Y. Yang, “Blockchain-based efficient privacy preserving and data sharing scheme of content-centric network in 5G,” IET Commun., vol. 12, no. 5, pp. 527–532, Mar. 2018.

[30] M. Singh and S. Kim, “Branch based blockchain technology in intelligent vehicle,” Compute. Newt., vol. 145, pp. 219–231, Nov. 2018.

[31] R. S. Da Silva and S. D. Zorzo, “An access control mechanism to ensure privacy in named data networking using attribute-based encryption with immediate revocation of privileges,” in Proc. 12th Annu. IEEE Consume. Commun. Newt. Conf., Jan. 2015, pp. 128–133.

[32] B. Li, D. Huang, Z. Wang, and Y. Zhu, “Attribute-based access control for ICN naming scheme,” IEEE Trans. Dependable Secure Compute., vol. 15, no. 2, pp. 194–206, Apr. 2016.