A

Major Project Report

on

Revocable Multi-Authority Attribute-Based Encryption with Time Based Authority

A report submitted in partial fulfillment of the requirements for

the award of degree of Bachelor of Technology

By

Kompella Venkat Sai Srujan

(20EG105304)

Jayanthi Nikhil Anand

(20EG105318)

Mahmood Ali Khan

(20EG105329)

Under The Guidance Of

P. Swarajya Lakshmi

Assistant Professor, CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ANURAG UNIVERSITY

VENKATAPUR (V), GHATKESAR (M), MEDCHAL (D), T.S - 500088

TELANGANA

(2020-2024)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Report/dissertation entitled "**Revocable Multi-Authority Attribute-Based Encryption with Time Based Authority"** that is being submitted by **Mr. Kompella Venkat Sai Srujan** bearing the hall ticket number **20EG105304**, **Mr. Jayanthi Nikhil Anand** bearing the hall ticket number **20EG105318**, **Mr. Mahmood Ali Khan** bearing the hall ticket number **20EG105329** in partial fulfillment for the award of **Bachelor of Technology** in **Computer Science and Engineering** to the **Anurag University** is a record of bonafide work carried out by him under my guidance and supervision for the academic year 2023 to 2024.

The results presented in this project have been verified and found to be satisfactory. The results embodied in this project report have not been submitted to any other University for the award of any other degree or diploma.

Signature of Supervisor
P. Swarajya Lakshmi
Assistant Professor
Department of CSE

Signature of the Dean
Dr. G. Vishnu Murthy
Dean,CSE

External Signature

DECLARATION

I hereby declare that the Report entitled "**Revocable Multi-Authority Attribute Based Encryption with Time Based Authority**" submitted for the award of Bachelor of technology Degree is my original work and the Report has not formed on the basis for the award of any degree, diploma, associate ship or fellowship of similar other titles. It has not been submitted to any other University or Institution for the award of any degree or diploma.

Place: Anurag University, Hyderabad

Kompella Venkat Sai Srujan
(20EG105304)

Jayanthi Nikhil Anand
(20EG105318)

Mahmood Ali Khan
(20EG105329)

ACKNOWLEDGEMENT

We would like to express our sincere thanks and deep sense of gratitude to project supervisor **P. Swarajya Lakshmi,** Assistant Professor for her constant encouragement and inspiring guidance without which this project could not have been completed. Her critical reviews and constructive comments improved my grasp of the subject and steered to the fruitful completion of the work. Her patience, guidance and encouragement made this project possible.

We would like to acknowledge our sincere gratitude for the support extended by **Dr. G. Vishnu Murthy**, Dean, Dept. of CSE, Anurag University. We also express my deep sense of gratitude to **Dr. V V S S S Balaram** , Academic coordinator, **Dr. T. Shyam Prasad,** Assistant Professor. Project Coordinator and Project review committee members, whose research expertise and commitment to the highest standards continuously motivated us during the crucial stage of our project work.

We would like to express my special thanks to **Dr. V. Vijaya Kumar**, Dean School of Engineering, Anurag University, for their encouragement and timely support in my B.Tech program.

Kompella Venkat Sai Srujan
(20EG105304)

Jayanthi Nikhil Anand
(20EG105318)

Mahmood Ali Khan
(20EG105329)

ABSTRACT

With the increasing reliance on cloud storage systems for managing sensitive data, the need for robust access control mechanisms has become paramount. This project introduces an innovative cryptographic solution, termed Revocable Multi-Authority Attribute-Based Encryption with Time-Based Authority. The proposed scheme addresses the challenges of secure access control, efficient user revocation, and decentralized attribute management by incorporating a time-based authority component into the multi-authority attribute-based encryption framework.The introduction of time-based authority enables dynamic management of access privileges over distinct periods. Each authority is granted temporal control over attributes, ensuring that access policies evolve with changing user roles or organizational requirements. This enhances the adaptability of the system to dynamic access control scenarios, offering a practical solution for organizations with evolving security and data management needs.This project contributes to the ongoing efforts to enhance the security infrastructure of cloud storage systems, offering a forward-looking approach to access control in dynamic and collaborative environments. The proposed scheme presents a valuable contribution for researchers and practitioners seeking advanced solutions to secure data in the evolving landscape of cloud computing.

Keywords: Cloud Computing, Attribute-based encryption, Multi-authority.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	**Full Name**
PKI	Public-Key Infrastructure
MFA	Multi-Factor Authentication
ABE	Attribute-Based Encryption
RABE-DI	Revocable Attribute-based encryption with data integrity
CP-ABE	Ciphertext policy Attribute-based encryption
CP-ABKS	Ciphertext-Policy Attribute-Based Keyword Search
KP-ABE	Key-Policy Attribute-Based Encryption
HIBE	Hierarchical Identity-Based Encryption
CPA	Chosen-plaintext Attack

1. INTRODUCTION

In the realm of cryptography, the secure exchange and sharing of data stand as paramount concerns, especially in an era defined by pervasive digital communication. The advent of sophisticated encryption techniques has significantly bolstered data security, enabling individuals, organizations, and systems to safeguard sensitive information from unauthorized access and interception. However, as data traverses networks and systems, the challenge of ensuring its confidentiality, integrity, and authenticity persists.

Data sharing in cryptography encompasses a multifaceted approach aimed at protecting information throughout its lifecycle – from creation and transmission to storage and retrieval. At its core, cryptography employs mathematical algorithms and protocols to encode plaintext data into ciphertext, rendering it unintelligible to anyone without the corresponding decryption key. This process forms the cornerstone of secure communication channels, facilitating confidential exchanges over public networks while mitigating the risk of eavesdropping and tampering.

Moreover, cryptography extends beyond mere encryption to encompass digital signatures, authentication mechanisms, and key management protocols, each playing a vital role in fostering trust and accountability in data sharing environments. Digital signatures, for instance, enable entities to verify the authenticity and integrity of messages, thereby thwarting malicious attempts to impersonate or manipulate data. Similarly, robust authentication mechanisms, such as public-key infrastructure (PKI) and multi-factor authentication (MFA), bolster the assurance of users' identities and access privileges, bolstering overall security posture.

In recent years, advancements in cryptographic techniques, including homomorphic encryption, zero-knowledge proofs, and secure multiparty computation, have opened new frontiers in secure data sharing. These innovations enable computation on encrypted data without revealing its contents, paving the way for privacy-preserving data analytics, collaborative research, and decentralized systems. However, alongside these advancements come complex challenges related to scalability, interoperability, and regulatory compliance, underscoring the need for holistic approaches to cryptographic design and implementation.

As society continues to embrace digital transformation, the imperative for robust and resilient cryptographic solutions becomes increasingly pronounced. Whether in the realms of finance, healthcare, or national security, the ability to securely share and exchange data underpins innovation, efficiency, and trust in the digital age. Thus, the quest for cryptographic excellence remains a perpetual endeavor, driven by the collective pursuit of privacy, security, and freedom in an interconnected world.

Cloud storage has fundamentally transformed the landscape of data management by providing users with a virtual space to securely store their digital assets. Traditionally, storing large amounts of data required physical hardware such as hard drives or servers, which often posed limitations in terms of accessibility, scalability, and reliability. However, cloud storage has addressed these challenges by offering an alternative solution that leverages the power of the internet and remote servers.

One of the most significant advantages of cloud storage is its accessibility. Users can access their stored data from anywhere with an internet connection, eliminating the need to carry physical storage devices or be tied to a specific location. Whether at home, in the office, or on the go, individuals and businesses can easily retrieve, upload, or modify their files using various devices such as computers, smartphones, or tablets.

Moreover, cloud storage solutions typically offer features such as automatic syncing, which ensures that files are consistently updated across all linked devices. This seamless synchronization mechanism enables users to work on documents collaboratively or access the latest version of a file without manual intervention. Additionally, many cloud storage providers offer file sharing capabilities, allowing users to easily share documents, photos, or videos with colleagues, clients, or friends via secure links or shared folders.

Another key benefit of cloud storage is its scalability. Unlike traditional storage solutions that require physical upgrades or replacements to expand storage capacity, cloud storage allows users to scale their storage space dynamically based on their needs. Whether an individual requires additional storage for personal files or a business needs to accommodate growing volumes of data, cloud storage providers offer flexible plans that can be easily adjusted to meet changing requirements.

Despite these advantages, concerns about data security and privacy remain paramount for many users. Recognizing this, cloud storage providers have invested heavily in implementing robust security measures, including encryption techniques to protect data both in transit and at rest. Additionally, authentication mechanisms such as multi-factor authentication (MFA) add an extra layer of security to prevent unauthorized access to user accounts.

In conclusion, cloud storage has become an indispensable tool in the modern digital landscape, offering unparalleled convenience, efficiency, and reliability for storing, accessing, and managing data. By harnessing the power of remote servers and the internet, cloud storage has revolutionized the way individuals and businesses store and interact with their digital assets, paving the way for a more connected and collaborative future.Some common features of cloud storage services include:

- **Scalability**: Cloud storage allows users to scale their storage needs up or down as required without worrying about physical limitations.
- **Accessibility**: Users can access their files from anywhere with an internet connection, using various devices such as computers, smartphones, or tablets.
- **Data Backup and Redundancy**: Cloud storage providers typically offer redundant storage to ensure data availability and protection against data loss.

1.1 Overview

Revocable Multi-Authority Attribute-Based Encryption with Time-Based Authority is an innovative cryptographic scheme designed to provide robust and flexible access control over sensitive data. At its core lies Attribute-Based Encryption (ABE), a powerful paradigm where encryption and decryption are contingent upon the attributes possessed by users and the access policies embedded within ciphertexts. What sets this scheme apart is its incorporation of a multi-authority framework, distributing authority among several entities. This not only enhances scalability but also ensures decentralization and resilience. Crucially, the system integrates revocable access control, enabling the swift and secure revocation of user privileges when necessary, whether due to role changes or policy violations. Additionally, the introduction of time-based authority adds a temporal dimension to access management, allowing for dynamic adjustments based on evolving circumstances or time-sensitive requirements. This amalgamation of ABE, multi-authority structure, revocability, and temporal control establishes a comprehensive and adaptable solution for safeguarding sensitive data in a variety of contexts.

1.2 Problem Statement

Revocable Multi-Authority Attribute-Based Encryption with Time-Based Authority is an advanced cryptographic scheme that addresses the need for secure and flexible data access control in various applications. This encryption method combines the benefits of attribute-based encryption (ABE) and revocable multi-authority to provide a robust solution for managing access to sensitive information.While traditional attribute-based encryption and revocable multi-authority schemes offer valuable features, they come with certain limitations. In standard attribute-based encryption, scalability can be an issue as the number of attributes increases. Additionally, the revocation of access rights in

multi-authority schemes may suffer from inefficiencies and complexities. Time-based authorities, on the other hand, are not always well-integrated into existing systems, leading to potential synchronization challenges.

1.3 Problem Illustration

To illustrate the problems addressed by our proposed Revocable Multi-Authority Attribute-Based Encryption with Time-Based Authority , let's consider a scenario in a cloud-based healthcare system. In this system, patient records are stored securely, and access is granted based on attributes such as medical specialty, role, and time sensitivity.

Scenario:

Scalability Challenge:

- Issue: The healthcare system involves numerous attributes, including medical specialty, patient status, and treatment type. As the number of attributes grows, traditional attribute-based encryption struggles to efficiently manage and scale access policies.
- Illustration: A doctor specializing in multiple areas requires access to patient records based on various attributes. The sheer number of attributes associated with different medical specialties complicates the access control system, leading to performance bottlenecks and increased computational overhead.

Revocation Inefficiency:

- Issue: In a multi-authority setting, revoking access rights in real-time can be inefficient and prone to delays. This is particularly critical in healthcare scenarios where immediate access changes are crucial for patient privacy and security.
- Illustration: When a doctor resigns or changes their medical specialty, the traditional revocation process may take time to update. During this delay, the doctor might retain access to patient records that are no longer within their professional scope, posing a potential security risk.

Integration of Time-Based Authorities:

- Issue: Integrating time-based authorities seamlessly into existing encryption schemes is often a challenge. Failure to synchronize time-based access control can lead to discrepancies, jeopardizing the security and privacy of sensitive information.

- Illustration: A healthcare system may require time-sensitive access to certain patient records based on treatment plans. Without effective integration of time-based authorities, there may be difficulties ensuring that access privileges are granted or revoked at the correct times, compromising the system's overall reliability.

2. LITERATURE SURVEY

Cloud computing has been offering cost-effective storage solutions to personal and large scale enterprise applications A revocable attribute-based encryption with data integrity (RABE-DI) [1] system includes the following entities: the original data owner, the cloud server, an authority party, and the recipients. The authority center (e.g. the PKG) manages the security parameters and keys for the scheme. For example, the public parameters for the system and the private keys for the participants. The data owner controls the access of the shared data. To encrypt the data and upload it to the server, he encrypts the data with a ciphertext-policy attribute-based encryption.The cloud server stores the ciphertexts and executes the revocation operations. After receiving the ciphertext, a recipient decrypts the ciphertext (original or revoked ciphertext) and also can verify the correctness of the ciphertext.

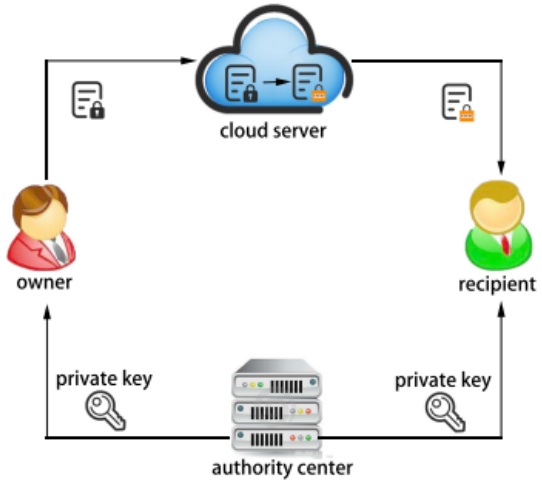


Fig 2.1 System Architecture

They present a multi-authority CP-ABE scheme [2] to support traceability with high expressiveness, and facilitate outsourcing decryption and updation of access policies. Our scheme is equipped with most of the essential requirements of practical attribute-based encryption systems and can be deployed in cloud applications.The advanced features of our proposed multi-authority cryptosystem are as follows:

1.Traceability: The malicious user who is involved in exposing decryption keys can be identified. White-box traceability is used with no identity table that improves the efficiency and requires zero storage overhead
2.Policy Updating: The data owner might require to modify access policy of ciphertext based on various requirements frequently. Such flexibility is provided by Policy updation, allowing fine-grained control to data owners to adjust the access policy over their encrypted data.
3.Outsourcing Decryption: The cost of decryption grows with the complexity of the access policy which limits users with less computational resources (for example in IoT devices). In such cases, the users can outsource costly decryption operation to more powerful computing resources and obtain a partially decrypted ciphertext.

The DCP-ABKS-CKDO [3] scheme introduces a groundbreaking approach to decentralized Conjunctive Predicate Attribute-Based Keyword Search (CP-ABKS) with the added functionality of outsourcing decryption for cloud computing environments. A notable feature of this scheme is its support for multi-keyword search, a capability previously unexplored in decentralized settings.One of the key advantages of theDCP-ABKS-CKDO scheme lies in its efficient online/offline encryption and outsourced decryption process. The offline encryption phase is independent of the plaintext message, encryption predicate, and keyword set. This means it can be executed at any time and even performed offline just once for different outsourced data. Such flexibility significantly enhances storage and computation efficiency, making it an attractive solution for cloud computing scenarios where resource optimization is crucial. The security of the scheme is rigorously formalized through definitions tailored to the decentralized CP-ABKS context. By proving its security in the random oracle model, the scheme demonstrates selective indistinguishability against chosen plaintext attacks (IND-CPA) and indistinguishability against chosen keyword attacks (IND-CKA). Furthermore, the scheme's design includes assigning a designated cloud server for search operations, rendering it immune to Key Generation (KG) attacks, thereby bolstering its resilience against potential security threats.To further enhance security, an extended variant of the scheme, termed DCP-ABKS-RC-CKDO, is proposed. This variant aims to achieve stronger protection against replayable chosen ciphertext attacks (RCCA), addressing additional security concerns and broadening the applicability of the scheme in environments where such threats may be prevalent.In summary, the DCP-ABKS-CKDO scheme presents a comprehensive solution for decentralized CP-ABKS with outsourced decryption, offering efficient operations, robust security guarantees, and flexibility in deployment, thereby addressing key challenges in cloud computing environments.

As more sensitive data is shared and stored by third-party sites on the Internet, there will be a need to encrypt data stored at these sites. One drawback of encrypting data, is that it can be selectively shared only at a coarse-grained level (i.e., giving another party your private key). They develop a new cryptosystem for fine-grained sharing of encrypted data that we call Key-Policy Attribute-Based Encryption (KP-ABE)[4]. In their cryptosystem, ciphertexts are labeled with sets of attributes and private keys are associated with access structures that control which ciphertexts a user is able to decrypt. They demonstrate the applicability of our construction to sharing of audit-log information and broadcast encryption. Their construction supports delegation of private keys which subsumes Hierarchical Identity-Based Encryption (HIBE).

Ciphertext-policy attribute-based encryption (CP-ABE) [5] is a promising cryptographic technique that integrates data encryption with access control for ensuring data security in IoT systems. However, the efficiency problem of CP-ABE is still a bottle neck limiting its development and application. A widespread consensus is that the computation overhead of bilinear pairing is excessive in the practical application of ABE, especially for the devices or the processors with limited computational resources and power supply. In this paper, they proposed a novel pairing-free data access control scheme based on CP-ABE using elliptic curve cryptography, abbreviated PF- CP-ABE. They replace complicated bilinear pairing with simple scalar multiplication on elliptic curves, thereby reducing the overall computation overhead. And they designed a new way of key distribution that it can directly revoke an user or an attribute without updating other users' keys during the attribute revocation phase. Besides, their scheme use linear secret sharing scheme (LSSS) access structure to enhance the expressiveness of the access policy. The security and performance analysis show that their scheme significantly improved the overall efficiency as well as ensured the security.

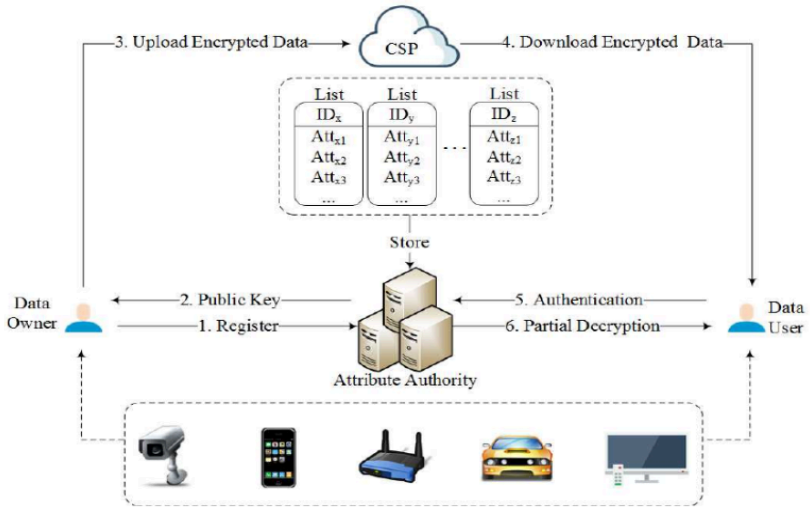


Fig 2.2 System Model

Cloud computing becomes increasingly popular for data owners to outsource their data to public cloud servers while allowing intended data users to retrieve these data stored in cloud. This kind of computing model brings challenges to the security and privacy of data stored in cloud. Attribute-based encryption (ABE) technology has been used to design fine-grained access control system, which provides one good method to solve the security issues in cloud setting. However, the computation cost and ciphertext size in most ABE schemes grow with the complexity of the access policy. Outsourced ABE (OABE) with fine-grained access control system [6] can largely reduce the computation cost for users who want to access encrypted data stored in cloud by outsourcing the heavy computation to cloud service provider (CSP). However, as the amount of encrypted files stored in cloud is becoming very huge, which will hinder efficient query processing. To deal with above problem, we present a new cryptographic primitive called attribute-based encryption scheme with outsourcing key-issuing and outsourcing decryption, which can implement keyword search function (KSF-OABE). The proposed KSF-OABE scheme is proved secure against chosen-plaintext attack (CPA). CSP performs partial decryption task delegated by data user without knowing anything about the plaintext. Moreover, the CSP can perform encrypted keyword search without knowing anything about the keywords embedded in trapdoor.

Attribute-Based Encryption (ABE) [7] offers a powerful mechanism for fine-grained access control over encrypted data, making it a promising solution for ensuring authorized data privacy in cloud storage environments. However, several challenges need to be addressed before ABE can be effectively deployed in practice.Firstly, the computational overhead associated with decryption can become prohibitively high as the complexity of access policies increases. To mitigate this issue, ABE schemes with outsourced decryption capabilities are preferred, as they offload the decryption burden from users, thereby enhancing efficiency and usability.Secondly, the need for attribute revocation arises when a user's attributes change, necessitating timely and effective adjustments to the user's access privileges. A practical ABE system should support attribute revocation mechanisms to ensure that access control remains aligned with the evolving attributes of users.Thirdly, the dynamic nature of access control policies demands ABE schemes capable of accommodating policy updates initiated by data owners. This requirement ensures that access control remains adaptable to changing organizational or regulatory requirements without compromising security or usability.In response to these challenges, a practical ABE scheme is proposed, addressing the aforementioned issues concurrently. This scheme not only supports outsourcing decryption to alleviate computational costs but also includes mechanisms for attribute revocation and policy updates. Furthermore, to enhance flexibility, the scheme accommodates a flexible number of attributes, achieves a large universe, and supports multiple attribute authorities.The proposed scheme's security and performance characteristics are thoroughly analyzed, followed by extensive experimental evaluations to demonstrate its effectiveness and practicability in real-world scenarios. By addressing the key challenges associated with ABE deployment, this scheme represents a significant advancement towards realizing secure and efficient data privacy management in cloud storage environments.

2.1 Comparison of existing Methods

Sl.No	Strategies	Advantages	Disadvantages
1	ABE scheme employs revocation (2021)	Users can share data without leaving traces or revealing their identities.	Designing and implementing such a scheme can be complex, requiring expertise in cryptography, secure systems design
2	Traceable multi-authority CP-ABE (2020)	Improve efficiency and expressiveness of our system.	This may potentially lead to exposure of decryption keys by some malicious users
3	A secure and efficient outsourced computation on data sharing scheme for privacy computing (2019)	Decentralization reduces reliance on a fully trusted central authority, making the system more robust	Outsourcing decryption to the cloud relies on trust in the cloud service provider.
4	Key-Policy Attribute-Based Encryption (2006)	Users can selectively share encrypted data based on specific attributes, providing a granular control over who can access the information	As the number of users and attributes increases, managing and scaling the system becomes more challenging
5	A novel efficient pairing-free CP-ABE based on elliptic curve cryptography for IoT (2018)	Reducing the overall computation overhead	Takes more time for computation
6	KSF-OABE: Outsourced attribute-based encryption (2017)	Secure against chosen-plaintext attack	Time taken for token exchange is more as search option is included
7	Efficient Attribute Revocation (2018)	Secure against collusion attack launched by the existing users and the revoked users	Works on only attribute based security

Table 2.1.1 Comparison of existing Methods

3.PROPOSED METHOD

3.1 Methodology

Revocable Multi-Authority Attribute-Based Encryption with Time-Based Authority represents a pioneering cryptographic framework that integrates attribute-based encryption (ABE) with multi-authority control and time-based revocation mechanisms. This innovative scheme addresses the complexities of access control in dynamic environments by allowing fine-grained access policies based on attributes, distributed authority management among multiple entities, and efficient revocation mechanisms. The introduction of time-based authority adds an additional layer of control, enabling access rights to be enforced only during specific time intervals, thereby enhancing security.
While traditional attribute-based encryption and revocable multi-authority schemes offer valuable features, they come with certain limitations. In standard attribute-based encryption, scalability can be an issue as the number of attributes increases. Additionally, the revocation of access rights in multi-authority schemes may suffer from inefficiencies and complexities. Time-based authorities, on the other hand, are not always well-integrated into existing systems, leading to potential synchronization challenges.

The proposed method, called Revocable Multi-Authority Attribute-Based Encryption with Time-Based Authority, introduces a new way to control access securely in cloud storage systems. Imagine a system where a central cloud server manages access, and different authorities are responsible for granting access based on specific attributes.

Now, the innovation here is the addition of a time-based authority, meaning that access privileges can change dynamically over time. A key advantage highlighted in the illustration is the efficient process of revoking access. This shows how access privileges can be smoothly adjusted in response to changing circumstances. The system's decentralized attribute management is also shown, with nodes working together to handle attributes. This highlights the system's scalability and resilience.

Revocable Multi-Authority Attribute-Based Encryption with Time-Based Authority is a cryptographic scheme that allows for fine-grained access control over encrypted data. Here's a general workflow of how it operates:

- **Setup Phase**:
    - **Key Generation**: A central authority generates master secret keys and public parameters.
    - **Attribute Authority Setup**: Multiple attribute authorities are set up, each responsible for managing specific attributes.

- **Key Distribution**:
    - Attribute-Based Key Generation: Users or entities request private keys based on their attributes from the respective attribute authorities.

- **Encryption**:
    - **Policy Specification**: The data owner specifies access policies in terms of attributes and time constraints.
    - **Encryption**: The data is encrypted under these policies using the public parameters.
- **Access Control**:
    - **Attribute Verification**: When a user or entity requests access to the encrypted data, their attributes are verified against the access policies.
    - **Time Verification:** The current time is checked against the time constraints specified in the access policies.
- **Key Retrieval**:
    - **Attribute Key Retrieval:** If the user's attributes match the access policies, they retrieve the corresponding attribute-based decryption key from the attribute authorities.

- **Decryption**:
    - **Decryption:** The user decrypts the encrypted data using the final decryption key.

- **Revocation**:
    - **Revocation:** If an entity's attributes are revoked, the corresponding decryption keys are updated or revoked by the attribute authorities.

- **Benefits:**
    - **Adaptability**: Time-based authority allows for the dynamic adjustment of access control policies.
    - **Efficient Revocation**: Users' access privileges can be revoked seamlessly based on the evolving time-based attributes.
    - **Decentralized Attribute Management**: Enhances scalability and resilience in attribute handling.

This illustration aims to provide a visual overview of the proposed method, showcasing its key features and demonstrating how it addresses the challenges of secure access control, efficient revocation, and decentralized attribute management in cloud storage systems.

3.2 Advantages of Proposed Method

- **Fine-Grained Access Control**: It allows for fine-grained access control over encrypted data. Users can define access policies based on attributes, and authorities can grant or revoke access accordingly.
- **Scalability**: The multi-authority aspect of improves scalability by distributing the authority among multiple entities. This can reduce the burden on a single authority and enhance system performance, especially in large-scale scenarios.
- **Flexibility**: Attribute-based encryption schemes in general offer flexibility in defining access policies based on various attributes of users and data. It extends this flexibility by introducing time-based authorities, enabling time-sensitive access control policies.
- **Revocability**: The revocable nature of the scheme allows authorities to revoke access privileges dynamically. This is particularly useful in scenarios where access rights need to be revoked due to policy changes, user revocation, or other reasons.
- **Security**: It aims to provide security against various cryptographic attacks while preserving the confidentiality of data. It employs cryptographic techniques to ensure that only authorized users can decrypt the data.

3.3 Disadvantages of Proposed Method

- **Complexity**: Implementing it can be complex due to its multi-authority nature and the incorporation of time-based attributes. This complexity may lead to challenges in system design, implementation, and management.
- **Key Management**: Managing cryptographic keys in a multi-authority environment can be challenging. Proper key management practices are essential to ensure the security and integrity of the system.
- **Performance Overhead**: The additional cryptographic operations required for enforcing fine-grained access control and revocation may introduce performance overhead, particularly in resource-constrained environments.
- **Potential Single Point of Failure**: While distributing authority among multiple entities enhances scalability, it also introduces the risk of a single point of failure if one or more authorities are compromised or unavailable.
- **Compatibility and Interoperability**: Integrating our method into existing systems may require modifications or customizations, and ensuring interoperability with other cryptographic schemes or protocols could be challenging.

In summary, while our method offers significant advantages in terms of access control, scalability, and flexibility, it also presents challenges related to complexity, key management, performance, security, and interoperability that need to be carefully addressed during implementation and deployment.

3.4 System Model

•**AAs**: AA, an independent attribute authority, which is an independent attribute authority that is responsible for entitling and revoking user's attributes according to their role or identity in its domain. The function of AA is generating public keys and secret keys, and distributing secret keys of attributes to data consumers. In addition, when one or more attributes are revoked, AA operates the update key generation algorithm for non-revoked data consumers.

•**CSP**: CSP is responsible for data storage. When an attribute is revoked, CSP will provide data access ser- vice to data consumers and perform a ciphertext update algorithm to help data owners update their ciphertext.

•**DO**: DO performs such operations as defining the access structure over attributes from one or more AAs, encrypting according to the access structure, and uploading
encrypted data to the CSP.

•**DC**: DC has access to ciphertexts from CSP, and requests their secret keys from corresponding AAs. Only if the DC's attribute set meets the access structure, DC decrypts the ciphertexts successfully with his secretkeys.

3.5 System architecture

These attributes play a crucial role in the encryption and access control process. Data is encrypted using Attribute-Based Encryption (ABE), incorporating attributes and access policies. This enables only users with matching attributes to decrypt and access the data. To maintain control over data access, a revocation mechanism is integrated, managed by a dedicated revocation authority. This allows for the prompt revocation of user attributes when necessary. Collaboration among multiple attribute authorities ensures the distribution of attributes and access policies, enhancing security through decentralization. The encrypted data is stored in the cloud, and authorized users can retrieve and decrypt it based on their attributes. The architecture also incorporates key updates to accommodate attribute changes. By combining ABE, attribute revocation, and multi-authority collaboration, the system aims to achieve efficient and secure data access control in cloud storage environments.

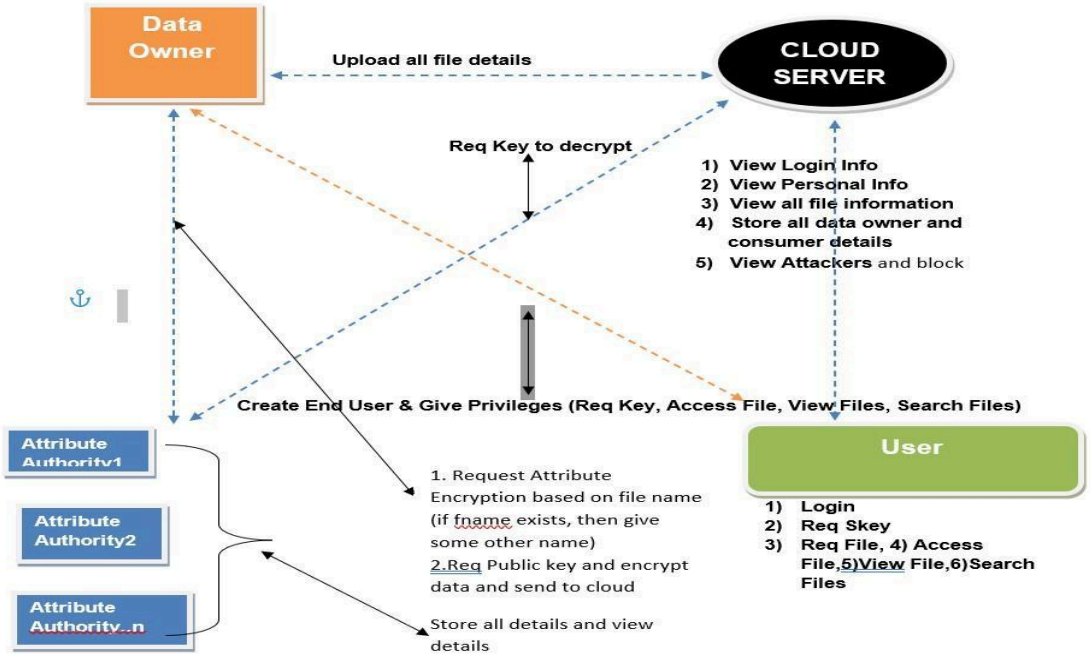


Fig 3.5.1 Proposed System Architecture

4. DESIGN

4.1 UML Diagrams

UML, or Unified Modeling Language, is a graphical tool essential for designing software systems. It offers standardized visual models for representing object-oriented software structures. UML diagrams are crucial for clear and organized communication of design concepts. These diagrams are indispensable in software design, aiding developers in understanding and analyzing intricate systems. They serve as effective tools for conveying design ideas to team members, stakeholders, and clients, ensuring that the software meets required standards of functionality, performance, and quality. Moreover, UML diagrams help in detecting and rectifying errors early in the development process, thereby saving time and reducing costs. They come in two main types: structural diagrams, which depict the static structure of the system, and behavioral diagrams, which illustrate dynamic interactions and Workflows. In summary, UML diagrams play a vital role in streamlining the software development process, providing developers with a clear and efficient means of conceptualizing and refining software systems.

4.1.1 Use Case Diagram

A use case diagram is a type of UML diagram that provides a visual representation of the interactions between users and a system. It illustrates the various use cases, or functionalities, of a system and the actors, or users, involved in those interactions. In a use case diagram, actors are depicted as stick figures, while use cases are represented as ovals. Arrows indicate the direction of interaction between actors and use cases. The diagram helps stakeholders understand how users interact with the system and the different functionalities the system offers. Use case diagrams are valuable tools for capturing and communicating the functional requirements of a system, facilitating discussions among stakeholders, and serving as a blueprint for system design and development. They provide a high-level overview of the system's behavior and help ensure that the system meets the needs and expectations of its uses.

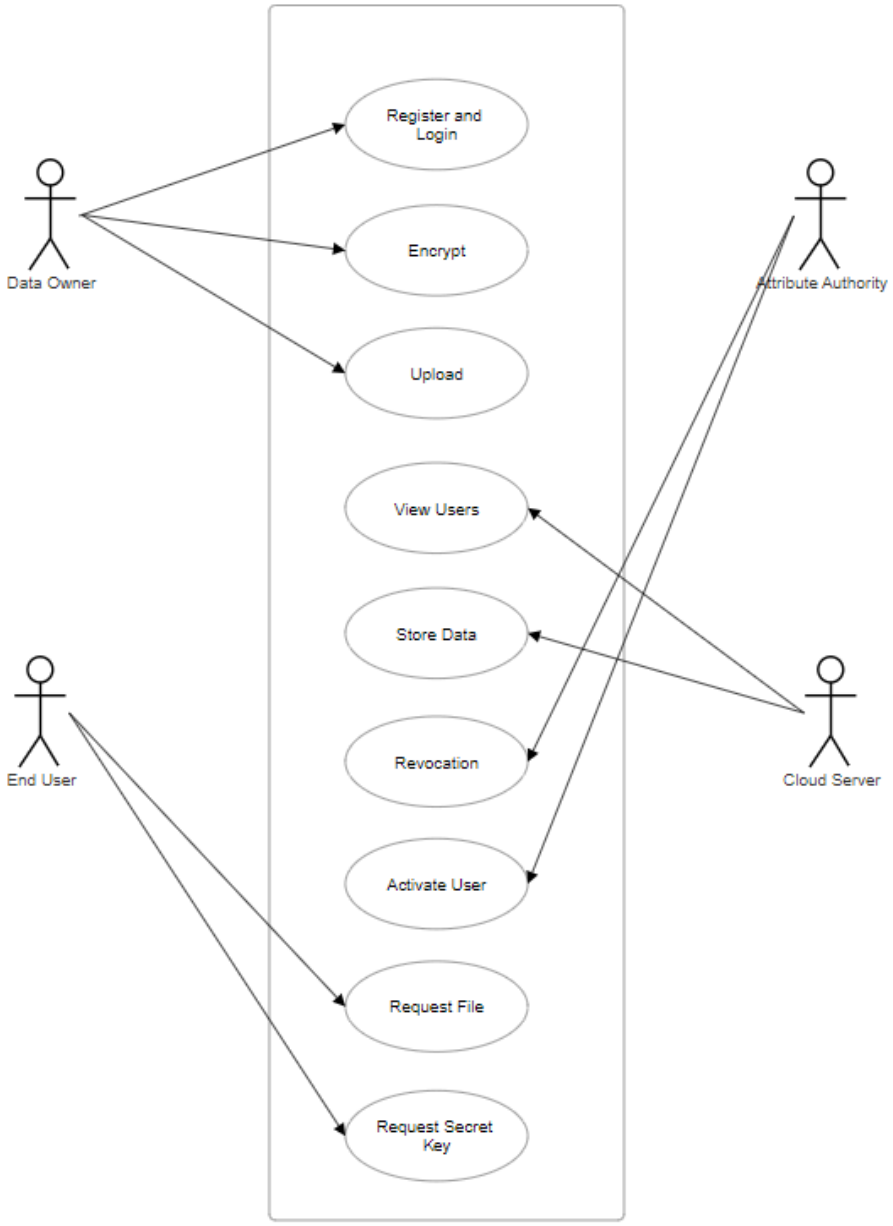


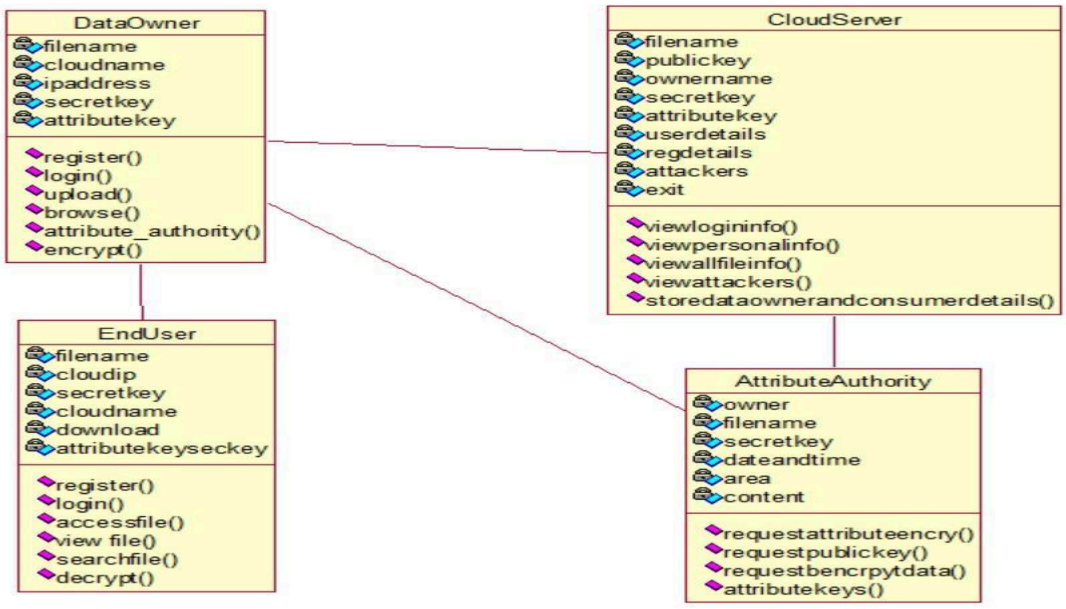
Fig 4.1.1.1 Use Case Diagram

The use case diagram provided illustrates the interactions and functionalities of a system implementing "Revocable Multi-Authority Attribute-Based Encryption with Time Based Authority." Below is a description of the components and their roles within the system:

1. **Data Owner:**
    - **Encrypt Data**: The user can encrypt sensitive data using the attributes and policies defined in the system.
    - **Upload Data**: Allows users to upload new data or files into the system.
    - **Register and Login**: This functionality allows new users to create accounts within the system. It also allows registered users to authenticate themselves and access the system.
2. **Cloud Service Provider:**
    - **Store Data:** It allows users to store encrypted data in the cloud
    - **View Users:** Users now not only access cloud services, manage resources, and monitor usage but also view users present in the system.
3. **End User:**
    - **RequestFile:** End-users can download files from the cloud storage service to their local devices.
    - **Request Secret Key:** End-users can request Attribute Authority for decryption key to access the contents of the encrypted file.
4. **Attribute Authority:**
    - **Revocation**: The Attribute Authority can initiate the revocation of a user's access privileges.
    - **Activate**:The Attribute Authority can activate or reactivate a user's access privileges within the system. This action might be necessary after a user's privileges were temporarily suspended or revoked and are now being reinstated.

4.1.2 Class Diagram

A class diagram serves as a visual blueprint for software systems, outlining the structure and connections between different components. In simple terms, it depicts the building blocks of a program, known as classes, and how they interact with each other. Each class represents a specific entity or object in the system, detailing its attributes (characteristics) and methods (actions). Associations between classes indicate relationships, illustrating how they collaborate to achieve system functionality. These diagrams are crucial for software development, providing a clear overview of the system's architecture, aiding in communication among developers, and facilitating the design and maintenance of complex software projects.



In this class diagram:

- EndUser: Represents the end user who interacts with the system. They have methods for logging in, downloading,viewing usage statistics.
- CloudServer: Represents the cloud server responsible for storing and retrieving data. It has methods for storing and retrieving data.
- DataOwner: Represents the entity who owns the data stored in the cloud server.
- AttributeAuthority: Represents the authority responsible for managing attributes related to access control. They have a method for revoking access to the data.

The relationships between these classes depict how they interact within the system. The end user utilizes the cloud server for storing and retrieving data. The data owner manages access to the data stored on the cloud server, and the attribute authority manages the attributes related to access control.

4.1.3 Sequence Diagram

A sequence diagram is a visual representation that illustrates the interactions and communication flow between different components or objects in a system over time. It demonstrates how these entities collaborate and exchange messages to achieve specific functionalities. In a sequence diagram, the vertical axis typically represents time, while horizontal lines, called lifelines, represent individual entities or objects involved in the interaction. Arrows between lifelines indicate the flow of messages or method calls between these entities, depicting the sequence of actions and responses. Sequence diagrams are invaluable tools in software engineering for designing and understanding the behavior of systems, as they provide a clear and intuitive visualization of the runtime interactions between various components. They aid in identifying potential bottlenecks, understanding the order of operations, and ensuring that system requirements are met. Additionally, sequence diagrams facilitate communication among stakeholders by offering a concise and structured depiction of system behavior, enabling effective collaboration and decision-making during the software development process. Overall, sequence diagrams play a crucial role in analyzing, designing, and documenting the dynamic behavior of complex systems.

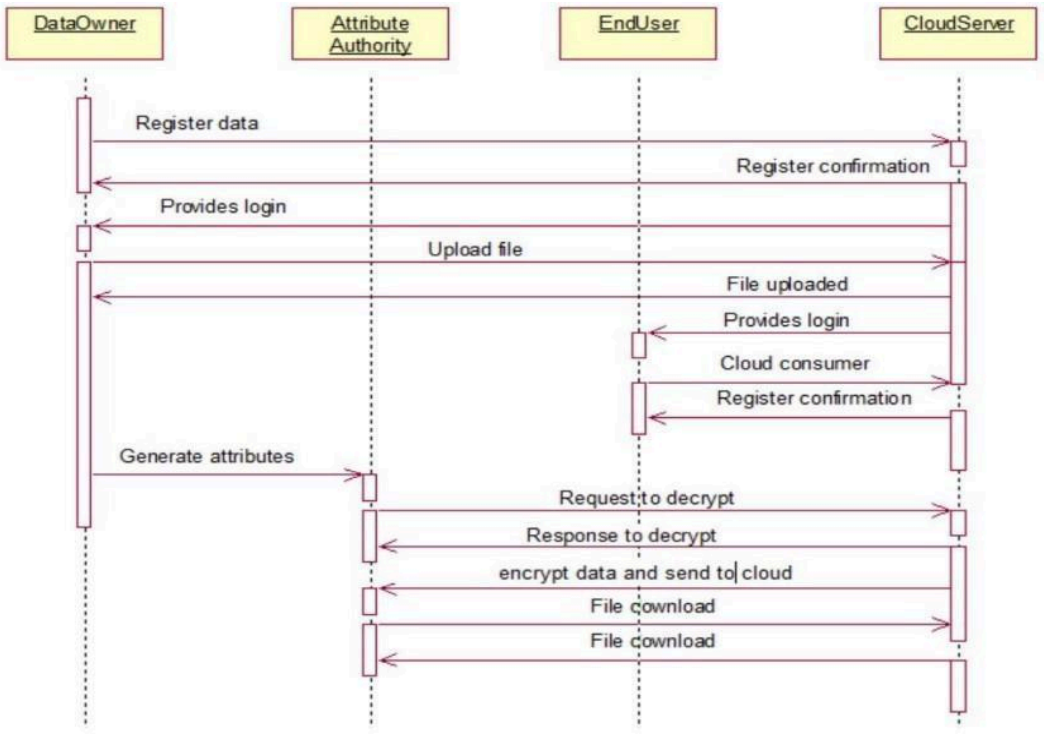


Fig 4.1.3.1 Sequence Diagram

This sequence diagram illustrates the interactions between the data owner, end user, cloud server, and attribute authority within a system. It shows how the data owner can upload files, how end users interact with the cloud server to download and upload files, and how the attribute authority manages access control settings in response to user requests.

4.1.4 Collaboration Diagram

A collaboration diagram, also known as a communication diagram, is a type of interaction diagram in the Unified Modeling Language (UML) that illustrates the interactions and relationships between objects or components within a system to achieve a specific functionality or behavior.

In a collaboration diagram, the emphasis is placed on how objects or components interact with each other to accomplish a task or respond to an event. It typically represents these interactions through messages exchanged between objects, which are depicted as labeled arrows between the objects. These messages can represent method calls, signals, or other forms of communication.

The main purpose of a collaboration diagram is to visualize the dynamic aspects of a system's behavior, showing how objects collaborate to fulfill certain requirements or achieve specific objectives. By depicting the flow of messages between objects, developers can gain insights into the runtime behavior of the system and identify potential design flaws or areas for optimization.

Overall, collaboration diagrams serve as valuable tools for understanding and communicating the runtime behavior and interactions within a system, aiding in system design, analysis, and debugging processes.

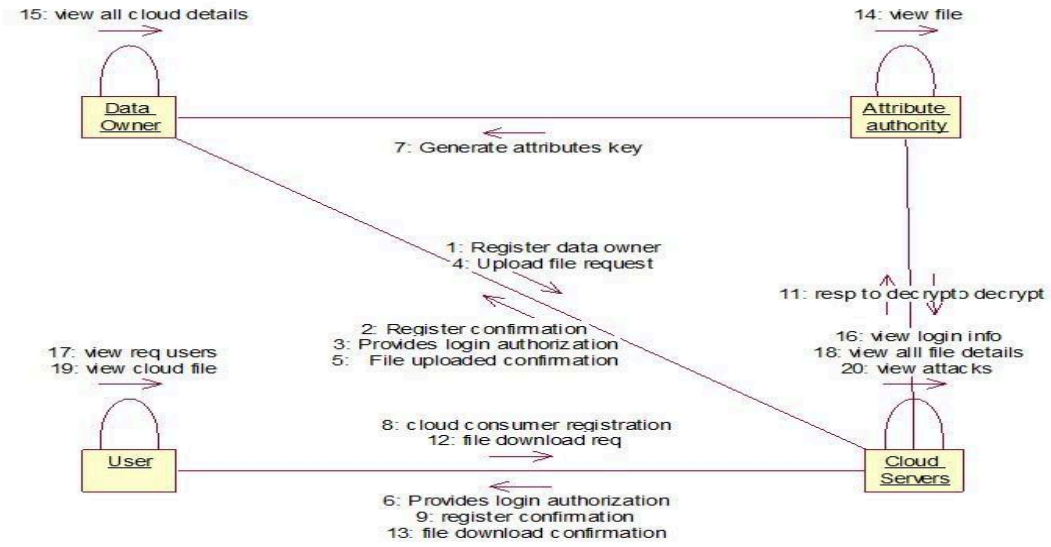


Fig 4.1.4.1 Collaboration Diagram

4.1.5 State Diagram

A state diagram, also known as a state machine diagram or statechart diagram, is a graphical representation used to depict the various states of an object or system and the transitions between those states. It is a behavioral diagram that shows the different states an object can be in over time, as well as the events that cause transitions between these states.

In a state diagram, each state is represented as a node, typically drawn as a rounded rectangle, and transitions between states are represented by arrows, showing the flow from one state to another. Events trigger these transitions, and conditions or actions associated with transitions may also be depicted. Additionally, the diagram may include initial and final states to denote the starting and ending points of the system's behavior.

State diagrams are commonly used in software engineering to model the behavior of systems, particularly those with complex state-dependent logic, such as user interfaces, control systems, or protocols. They provide a visual representation that helps developers understand and analyze the behavior of the system, aiding in the design, implementation, and testing phases of software development.

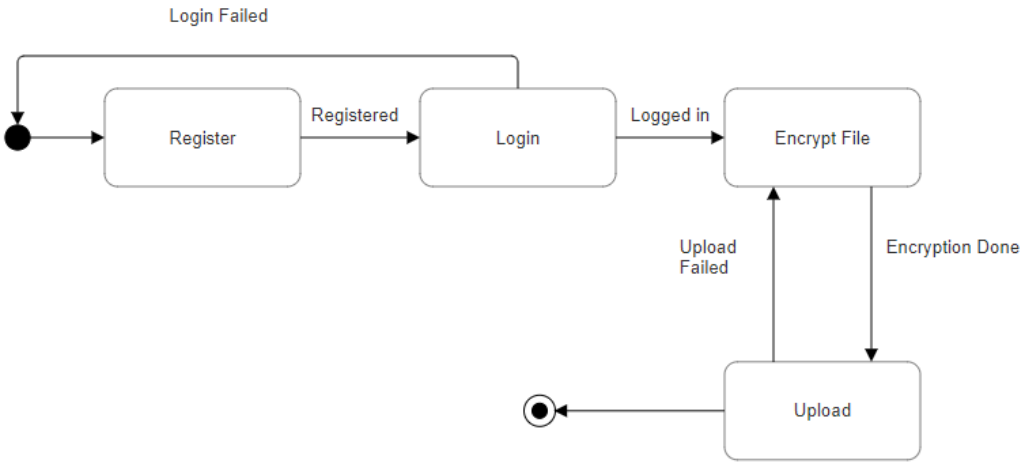


Fig 4.1.5.1 Data Owner State Diagram

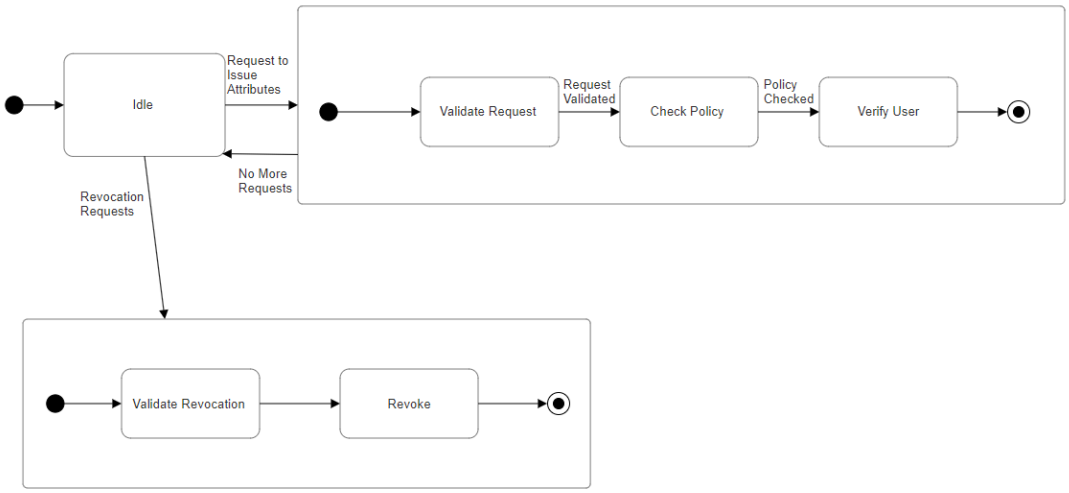


Fig 4.1.5.2 Attribute Authority State Diagram

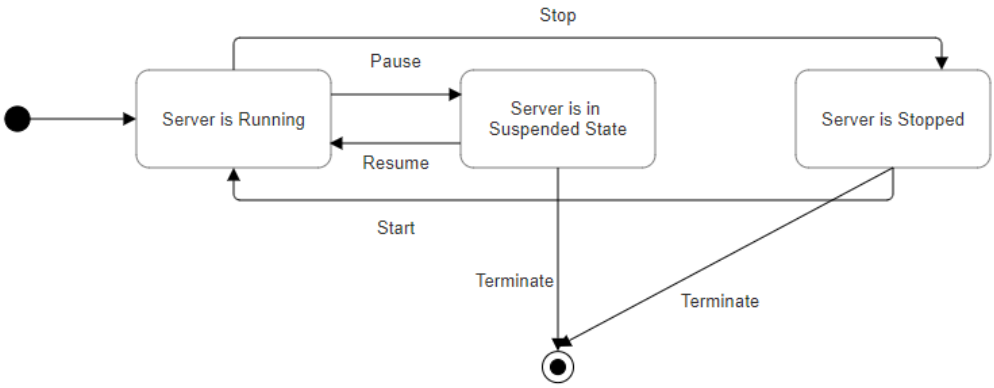


Fig 4.1.5.3 Cloud Server State Diagram

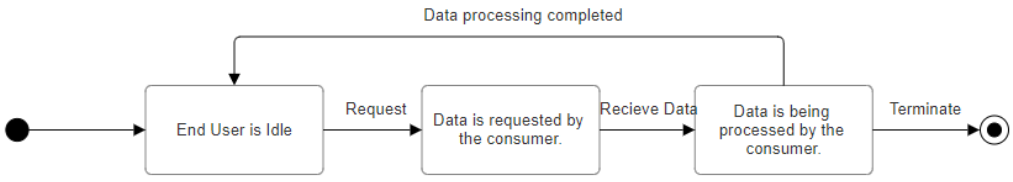


Fig 4.1.5.4 End User State Diagram

4.1.6 Activity Diagram

An activity diagram is a type of UML (Unified Modeling Language) diagram used to visually represent the flow of control or the sequence of activities in a system, process, or workflow. It typically consists of various activities, represented by nodes, connected by arrows to indicate the sequence in which the activities are performed. Each activity can represent a single operation, a group of operations, or a decision point within the system. The arrows between activities show the order in which the activities are executed, with forks and joins indicating parallel or concurrent execution paths. Activity diagrams are useful for modeling complex business processes, software workflows, or any sequence of actions that involve multiple actors or components. They provide a clear and concise visualization of how the system functions, helping stakeholders to understand, analyze, and communicate the behavior and logic of the system effectively.

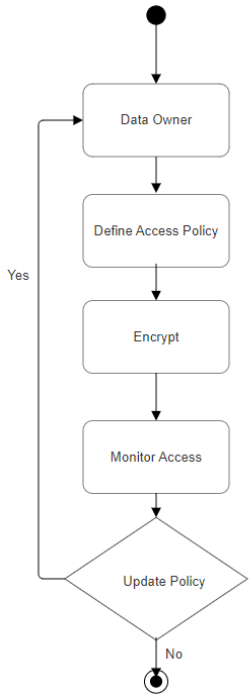


Fig 4.1.6.1 Data Owner Activity Diagram

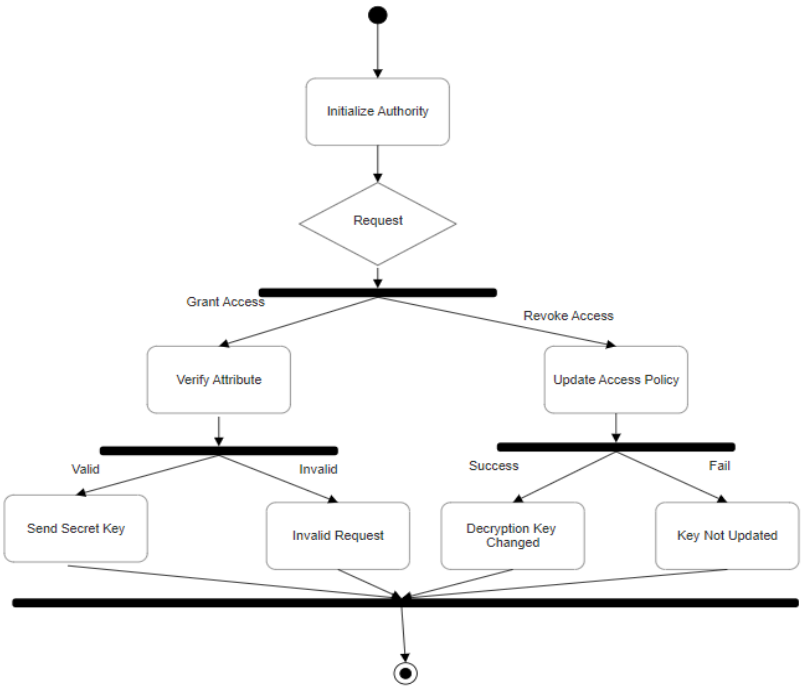


Fig 4.1.6.2 Attribute Authority Activity Diagram

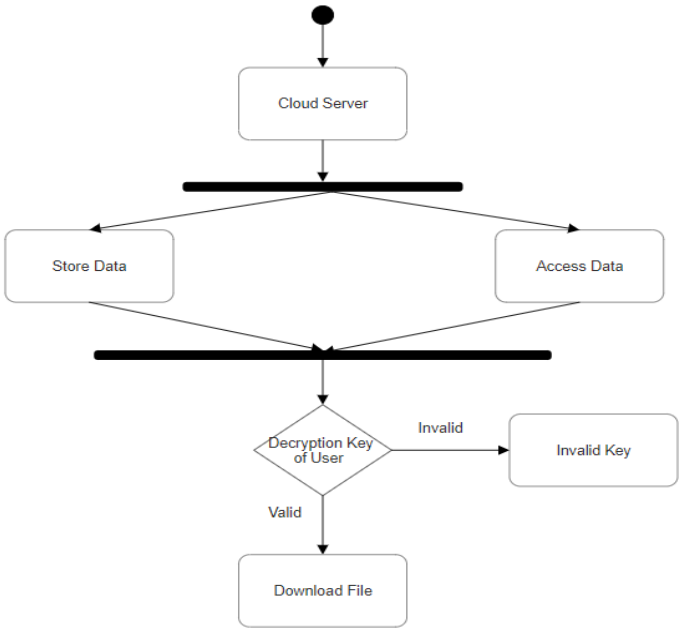


Fig 4.1.6.3 Cloud Server Activity Diagram

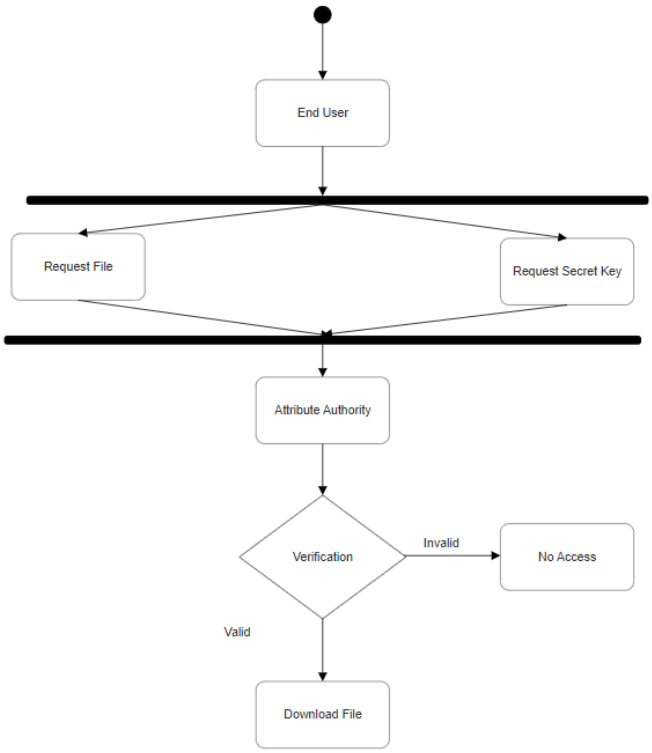


Fig 4.1.6.4 End User Activity Diagram

4.1.7 Component Diagram

A component diagram is a type of UML (Unified Modeling Language) diagram that depicts the components of a system and their relationships. It illustrates how various software components or modules interact with each other to form a coherent system architecture. Components are represented as rectangles, each encapsulating the functionality they provide. Dependencies between components are depicted using arrows, indicating the direction of communication or dependency. Typically, a component diagram is used during the design phase of software development to visualize the high-level structure of the system and its constituent parts. It helps software architects and developers to understand the organization of the system, identify dependencies between components, and plan for integration and deployment. Additionally, component diagrams facilitate communication among stakeholders by providing a clear representation of the system's architecture and its components.

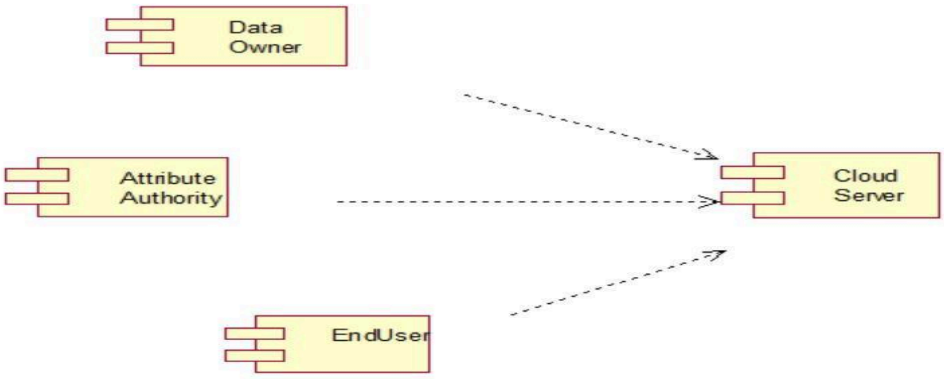


Fig 4.1.7.1 Component Diagram

4.1.8 Deployment Diagram

A deployment diagram is a type of UML (Unified Modeling Language) diagram that illustrates the physical deployment of software components within a computing environment. It shows how software artifacts, such as executable files, libraries, and databases, are distributed across hardware nodes, such as servers, workstations, or devices. Nodes are represented as boxes, while artifacts are depicted as rectangles, often connected by deployment relationships indicating which artifacts are hosted or executed on which nodes. Deployment diagrams are particularly useful for understanding the configuration and arrangement of software and hardware elements in a distributed system, including networks, servers, routers, and other infrastructure components. They help system architects and developers to plan for deployment, scalability, and performance optimization by visualizing how software components are distributed across different nodes and how they interact with each other over a network. Deployment diagrams also facilitate communication among stakeholders by providing a clear representation of the system's physical architecture and deployment topology.

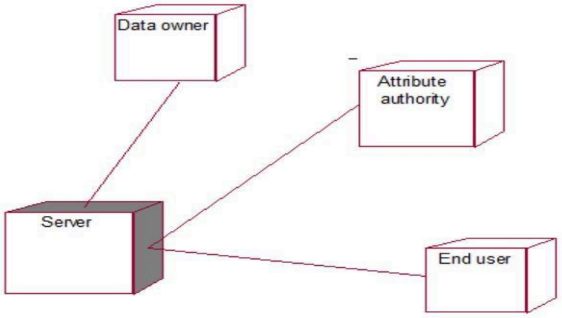


Fig 4.1.8.1 Deployment Diagram

5. IMPLEMENTATION

5.1 Packages and Libraries:

- **java.io**: The java.io package in Java is fundamental for handling input and output (I/O) operations within Java programs. It encompasses a comprehensive set of classes and interfaces tailored for various I/O tasks, including reading from and writing to streams, files, and other sources. At its core are abstract classes like InputStream and OutputStream, which serve as the foundation for all stream-based I/O operations. These classes are extended by concrete implementations such as FileInputStream and FileOutputStream, which facilitate interaction with files in the file system. The package also offers buffering capabilities through classes like BufferedInputStream and BufferedOutputStream, which enhance performance by reducing the frequency of I/O operations.

- **java.sql**: The java.sql package in Java provides a standardized interface for database access. It serves as a crucial component for developing database-driven applications by offering a consistent API for interacting with relational databases. At its core is the Connection interface, which represents a connection to a database and provides methods for executing SQL statements, managing transactions, and handling database metadata. Alongside Connection, the package includes interfaces like Statement and PreparedStatement, which allow developers to execute SQL queries and updates against the database.

- **javax.crypto**: The javax.crypto package in Java provides comprehensive support for cryptographic operations, offering developers a set of classes and interfaces to implement encryption, decryption, key management, and other cryptographic functionalities within Java applications. At its core, the package encompasses key classes such as Cipher, which encapsulates encryption and decryption algorithms, and SecretKey, which represents secret cryptographic keys used in symmetric encryption algorithms. These classes facilitate the encryption and decryption of data, ensuring its confidentiality and integrity during transmission or storage. Additionally, the package includes classes like KeyGenerator and KeyPairGenerator for generating cryptographic keys, enabling secure key management practices.

- **javax.servlet**: The javax.servlet package in Java provides a foundational framework for developing web applications using the Servlet API. Servlets are Java classes that dynamically process and respond to client requests sent over HTTP. At the core of this package lies the Servlet interface, which defines methods for initializing a servlet, processing requests, and generating responses. Servlets encapsulate business logic and interact with the request and response objects provided by the Servlet API to handle various web application functionalities.

- **org.apache.commons.fileupload**:The org.apache.commons.fileupload package is a component of the Apache Commons FileUpload library, which provides utilities for handling file uploads in web applications. This library simplifies the process of parsing HTTP requests containing file upload data and provides a convenient API for processing uploaded files within Java servlets or other web frameworks.At the core of the org.apache.commons.fileupload package is the FileUpload class, which offers methods for parsing and processing file upload requests. It abstracts away the complexities of handling multipart/form-data requests, which are commonly used for file uploads in web applications.

- **com.sun.org.apache.xerces.internal.impl.dv.util**:
  The com.sun.org.apache. xerces.internal.impl.dv.util package is part of the Xerces XML parser implementation provided by the Apache Software Foundation. Specifically, it includes utility classes related to data types and values used in XML Schema validation and processing.At the core of this package lies a set of utility classes designed to assist in the handling of XML Schema built-in data types (such as integers, strings, dates, etc.) and their corresponding values. These utility classes provide methods for parsing, formatting, and manipulating values conforming to XML Schema data types.

- **com.sun.org.apache.xerces.internal.impl.dv.util.Base64:**
  The com.sun.org.apache.xerces.internal.impl.dv.util.Base64 class is a part of the Xerces XML parser implementation provided by the Apache Software Foundation.Specifically,it's located within the package and serves as a utility for encoding and decoding data using the Base64 encoding scheme.Base64 encoding is commonly used in various contexts, especially in web applications, to represent binary data in an ASCII format that is safe for transmission across different systems, such as email or HTTP protocols. The Base64 class provides methods for encoding binary data into Base64-encoded strings and decoding Base64-encoded strings back into their original binary representation.

- j**ava.io.ByteArrayOutputStream:** The java.io.ByteArrayOutputStream class in Java provides a convenient way to write data to an in-memory byte array. It extends the OutputStream class, allowing bytes to be written to the underlying byte array buffer. This class is particularly useful when you need to capture output generated by various methods or write data to a byte array instead of a file or network stream.At its core, ByteArrayOutputStream maintains an internal buffer that automatically grows as data is written to it. It offers methods for writing bytes, such as write(int b) for writing a single byte and write(byte[] b, int off, int len) for writing a portion of an array of bytes.One of the key advantages of ByteArrayOutputStream is its simplicity and flexibility. Since it operates entirely in memory, it eliminates the need to manage file I/O operations or worry about resource cleanup associated with file streams. This makes it ideal for scenarios where temporary storage of data or efficient manipulation of byte data is required, such as in-memory data processing or generating byte arrays for network communication.

- **java.io.FileInputStream:** The java.io.FileInputStream class in Java provides a way to read data from a file in the file system as a stream of bytes. It extends the InputStream class, making it suitable for reading binary data from files. This class is commonly used in scenarios where reading data from a file is required, such as file processing, input for data analysis, or reading configuration files.At its core, FileInputStream opens a connection to the specified file, allowing bytes to be read from it sequentially. It offers methods for reading bytes, such as read() for reading a single byte and read(byte[] b, int off, int len) for reading bytes into an array of bytes.One of the main advantages of FileInputStream is its simplicity and straightforward usage. Developers can easily instantiate a FileInputStream object by providing the path to the file they want to read, and then use standard methods for reading data from the file.

- **java.io.FileWriter:** The java.io.FileWriter class in Java provides a convenient way to write character data to a file in the file system. It extends the Writer class, making it suitable for writing text data to files. This class is commonly used in scenarios where creating or appending text files is required, such as logging, generating reports, or saving user data.At its core, FileWriter opens a connection to the specified file, allowing characters to be written to it sequentially. It offers methods for writing characters, such as write(int c) for writing a single character and write(char[] cbuf, int off, int len) for writing characters from an array of characters.One of the main advantages of FileWriter is its simplicity and straightforward usage. **java.util.Scanner:** This package is used for scanning input from various sources.

- **javax.crypto.Cipher:**The javax.crypto.Cipher class in Java is a fundamental component of the Java Cryptography Architecture (JCA), providing an interface for encryption and decryption operations. This class is central to implementing cryptographic algorithms and ensuring data security within Java applications.At its core, Cipher provides methods for encrypting and decrypting data using various cryptographic algorithms, such as AES (Advanced Encryption Standard), DES (Data Encryption Standard), and RSA (Rivest-Shamir-Adleman). It supports both symmetric and asymmetric encryption techniques, allowing developers to choose the most suitable approach based on their security requirements.One of the primary advantages of Cipher is its flexibility and ease of use. Developers can instantiate a Cipher object and initialize it with a specific cryptographic algorithm and mode of operation (e.g., ECB, CBC, or GCM). They can then use the encrypt and decrypt methods to process input data and obtain encrypted or decrypted output.

- **javax.crypto.KeyGenerator:** The javax.crypto.KeyGenerator class in Java is a fundamental component of the Java Cryptography Architecture (JCA), providing functionality for generating symmetric cryptographic keys. This class is essential for ensuring the security of cryptographic operations by creating keys that are used in encryption and decryption processes.At its core, KeyGenerator offers methods for generating symmetric keys for various cryptographic algorithms, such as AES (Advanced Encryption Standard), DES (Data Encryption Standard), and Blowfish. These keys are typically used in symmetric encryption schemes, where the same key is used for both encryption and decryption.One of the primary advantages of KeyGenerator is its simplicity and ease of use. Developers can instantiate a KeyGenerator object and initialize it with a specific cryptographic algorithm. They can then use the generateKey method to generate a new symmetric key of the specified algorithm and key size.

- **javax.crypto.SecretKey:** The javax.crypto.SecretKey interface in Java is a fundamental component of the Java Cryptography Architecture (JCA), representing a secret (symmetric) cryptographic key. This interface serves as a key abstraction for symmetric encryption algorithms, facilitating secure communication and data protection within Java applications.At its core, SecretKey encapsulates the secret key material used for encryption and decryption operations in symmetric cryptography. It abstracts away the details of key management and provides a standardized interface for working with secret keys across different cryptographic algorithms.One of the primary advantages of SecretKey is its flexibility and interoperability. It allows developers to work with secret keys in a uniform manner, regardless of the underlying cryptographic

algorithm used for encryption. This simplifies the implementation of cryptographic operations and enhances code maintainability.

- **javax.crypto.spec.SecretKeySpec:** The javax.crypto.spec.SecretKeySpec class in Java is a fundamental component of the Java Cryptography Architecture (JCA), providing a way to create a secret (symmetric) cryptographic key from raw key bytes. This class allows developers to specify the key material directly as an array of bytes, enabling the creation of secret keys from pre-existing key material or from external sources.At its core, SecretKeySpec encapsulates the key material as an array of bytes and associates it with a specific cryptographic algorithm. This allows developers to create secret keys for various symmetric encryption algorithms, such as AES (Advanced Encryption Standard), DES (Data Encryption Standard), and Blowfish.One of the primary advantages of SecretKeySpec is its simplicity and ease of use. Developers can instantiate a SecretKeySpec object by providing the key material as an array of bytes and the name of the cryptographic algorithm. This allows for straightforward creation of secret keys without the need for complex key generation processes.

- **javax.swing.JOptionPane:** The javax.swing.JOptionPane class in Java provides a simple yet versatile way to display dialog boxes for interacting with users in graphical user interface (GUI) applications. It is part of the Swing framework, which is used for building desktop applications with Java.At its core, JOptionPane allows developers to create and display various types of dialog boxes, including message dialogs, input dialogs, confirm dialogs, option dialogs, and more. These dialog boxes can be used to convey information, prompt users for input, or confirm actions.One of the main advantages of JOptionPane is its ease of use and flexibility. Developers can easily create a dialog box by calling static methods of JOptionPane, passing parameters such as the message to be displayed, the type of dialog box, and the available options. This makes it straightforward to incorporate user interaction into GUI applications without the need for creating custom dialog components.
- **sun.misc.BASE64Decoder:** The sun.misc.BASE64Decoder class in Java is part of the sun.misc package, which contains various miscellaneous and unsupported classes provided by the Sun Microsystems (now Oracle Corporation). This particular class is used for decoding Base64-encoded data into its original binary form.At its core, sun.misc.BASE64Decoder offers functionality to decode data encoded using the Base64 encoding scheme. Base64 encoding is commonly used to represent binary data as ASCII characters, making it suitable for transmitting binary data through channels that may not support binary data directly, such as email or HTTP headers.Despite its functionality, it's important to note that classes

within the sun.misc package are not part of the public API and are subject to change or removal without notice. These classes are considered internal to the Java platform and are not intended for direct use by developers. As such, using classes from the sun.misc package in production code is generally discouraged, as they may not be supported in future Java releases and may have limited compatibility across different Java implementations.

- **sun.misc.BASE64Encoder:** The sun.misc.BASE64Encoder class in Java is part of the sun.misc package, which contains various miscellaneous and unsupported classes provided by Sun Microsystems (now Oracle Corporation). This class is utilized for encoding binary data into Base64 format, allowing binary data to be represented as ASCII characters, which is useful for transmitting binary data through channels that may not support binary data directly, such as email or HTTP headers.At its core, sun.misc.BASE64Encoder offers functionality to encode binary data using the Base64 encoding scheme. This encoding process involves breaking the input binary data into groups of 3 bytes, converting each group into four 6-bit values, and then representing each 6-bit value as a printable ASCII character from the Base64 alphabet.

5.2 Attributes

- **filename**: Represents the name of the uploaded file.
- **address**: Represents an address related to the uploaded file.
- **longi**: Represents longitude information associated with the uploaded file.
- **lati**: Represents latitude information associated with the uploaded file.
- **protocol**: Represents a protocol related to the uploaded file.
- **quality1**: Appears to be commented out, but possibly represents some quality information related to the file.
- **email:** Represents the email associated with the uploaded file, presumably the email of the user performing the upload.
- **date**: Appears to be commented out, but possibly represents a date associated with the file upload.
- **decrypted text**: This variable stores the decrypted text obtained after decryption.
- **txt**: This variable represents the encrypted text passed to the decrypt method as a parameter.
- **skey**: This variable represents the secret key passed to the decrypt method as a parameter.
- **bs**: This variable stores the byte array obtained after decoding the Base64-encoded secret key.
- **sec**: This variable represents the SecretKey object obtained from the decoded byte array.
- **aesCipher**: This variable represents the Cipher object used for encryption and decryption using the AES algorithm.
- **byteCipherText**: This variable stores the byte array obtained after decoding the Base64-encoded encrypted text.
- **byteDecryptedText**: This variable stores the byte array obtained after decryption of the encrypted text.
- **plainData**: This variable is intended to hold the plaintext input before encryption.
- **cipherText**: This variable is intended to hold the encrypted ciphertext after encryption.

5.3 Variables

- **co**: Represents a database connection (Connection object).
- **pstm**: Represents a prepared statement (PreparedStatement object) for database operations.
- **filename**: Stores the name of the uploaded file (String).
- **address**: Stores an address related to the uploaded file (String).
- **protocol**: Stores a protocol related to the uploaded file (String).
- **quality1:** Appears to be commented out, possibly intended to store some quality information related to the file (String).
- **email:** Stores the email associated with the uploaded file, presumably the email of the user performing the upload (String).
- **isMultipartContent:** Indicates whether the request contains multipart content (boolean).
- **factory:** Represents a file item factory (FileItemFactory object) for processing file uploads.
- **upload:** Represents a file upload handler (ServletFileUpload object).
- **fields:** Represents a list of file items parsed from the request (List<FileItem>).
- **it:** Iterator for iterating over file items (Iterator<FileItem>).
- **fileItem:** Represents a file item in the list (FileItem object).
- **isFormField:** Indicates whether the file item represents a form field (boolean).
- **str:** Represents the content of the uploaded file as a String.
- **keyGen:** Represents a key generator for generating encryption keys (KeyGenerator object).
- **secretKey:** Represents a secret key for encryption (SecretKey object).
- **be:** Represents the encoded form of the secret key (byte[]).
- **skey**: Represents the secret key encoded as a Base64 string (String).
- **cipher:** Represents the encrypted content of the file (String).
- **pst2:** Represents a prepared statement (PreparedStatement object) for inserting data into the database.
- **plainData:** Stores the plaintext input before encryption.
- **cipherText:** Stores the encrypted ciphertext after encryption.
- **aesCipher:** Instance of Cipher class used for AES encryption.
- **byteDataToEncrypt:** Byte array representation of the plaintext data to be encrypted.
- **byteCipherText:** Byte array representation of the encrypted ciphertext.

5.4 Technology Used

5.4.1 Java

Java was developed by James Gosling and his team at Sun Microsystems (which was later acquired by Oracle Corporation) in the early 1990s. The first public release of Java, Java 1.0, occurred in 1996. It was designed with the principle of "Write Once, Run Anywhere" (WORA), meaning that Java programs can run on any platform that supports Java without needing to be recompiled. This was made possible by compiling Java source code into an intermediate bytecode, which can then be executed by the Java Virtual Machine (JVM).

One of Java's defining features is its full embrace of object-oriented programming (OOP) principles, enabling developers to build modular, scalable, and maintainable codebases. With automatic memory management via garbage collection, Java liberates programmers from the burdens of manual memory allocation and deallocation, enhancing productivity and reducing the likelihood of memory-related errors.
Java's strength lies in its rich standard library, encompassing a vast array of classes and methods catering to diverse application needs, from basic I/O operations to advanced networking and data manipulation. Furthermore, Java's robust exception handling mechanism bolsters the resilience of applications, ensuring graceful recovery from unexpected errors.

Beyond its robustness and flexibility, Java's security features, including bytecode verification and a built-in security manager, make it a preferred choice for developing secure systems. It founds a strong foothold in enterprise software development, powering critical systems in banking, e-commerce, and customer relationship management (CRM). Moreover, Java remains instrumental in web development, with frameworks like Spring and JavaServer Faces (JSF) enabling the creation of robust server-side applications and web services.

In the mobile realm, Java's influence persists through its use in Android app development, offering a familiar platform for crafting mobile experiences. Additionally, Java's versatility extends to desktop applications, where libraries like JavaFX and Swing facilitate the creation of graphical user interfaces (GUIs) for various operating systems.
Not limited to traditional domains, Java's adaptability finds application in emerging fields like embedded systems, where its portability and security features make it a compelling choice. Despite the evolution of programming landscapes, Java's enduring popularity, driven by its reliability, versatility, and extensive ecosystem, cements its position as a cornerstone of modern software development.

5.4.2 Java Server Pages

JavaServer Pages (JSP) stands as a crucial technology in the landscape of web development, empowering developers to craft dynamic web pages with Java seamlessly integrated within HTML. With JSP, developers embed Java code snippets within HTML markup, allowing for the creation of dynamic content that responds to user interactions and data from various sources. When a user accesses a JSP page, the server translates it into a Java servlet, which is then compiled and executed to produce the final HTML output. This process enables the generation of personalized content based on user input, database queries, or other dynamic factors.

One of the key strengths of JSP lies in its familiarity and ease of use. Developers can leverage their existing knowledge of HTML alongside Java to build feature-rich web applications without a steep learning curve. Additionally, JSP seamlessly integrates with other Java Enterprise Edition (EE) technologies such as Servlets, JDBC for database access, and JavaServer Pages Standard Tag Library (JSTL) for common tasks, facilitating the development of robust and scalable web solutions.

Furthermore, JSP promotes modular and reusable components through custom tags and JavaBeans, enabling developers to encapsulate functionality and enhance code maintainability. This modularity facilitates collaboration among development teams and streamlines the development process, leading to more efficient and scalable applications.
Despite its long-standing presence in the web development arena, JSP continues to evolve alongside modern technologies and frameworks. While alternatives such as Servlets with JAX-RS and front-end JavaScript frameworks have gained popularity for certain use cases, JSP remains a viable choice for projects requiring server-side rendering and close integration with Java EE ecosystems.

In conclusion, JavaServer Pages (JSP) remains a cornerstone of dynamic web development, offering a powerful combination of Java's robustness and HTML's simplicity. Its seamless integration with Java EE APIs, familiarity, and versatility make it a valuable tool for building dynamic and interactive web applications tailored to diverse requirements and environments.

5.4.3 Cryptography

Cryptography is a fundamental aspect of modern information security, dealing with the protection of sensitive data by converting it into a form that is unreadable to unauthorized parties. It encompasses a broad range of techniques, algorithms, and protocols designed to ensure the confidentiality, integrity, and authenticity of data.

At its core, cryptography involves the use of mathematical algorithms to transform plaintext (readable data) into ciphertext (unreadable data). This transformation process, known as encryption, typically relies on cryptographic keys to govern the encryption and decryption operations. The encrypted data can only be decrypted back into its original plaintext form by those who possess the corresponding decryption key.

Encryption algorithms vary in complexity and strength, ranging from symmetric encryption, where the same key is used for both encryption and decryption, to asymmetric encryption, where a pair of keys (public and private) is employed. Symmetric encryption algorithms include widely-used standards like AES (Advanced Encryption Standard) and DES (Data Encryption Standard), while asymmetric encryption algorithms include RSA (Rivest–Shamir–Adleman) and ECC (Elliptic Curve Cryptography).

In addition to encryption, cryptography encompasses various other cryptographic techniques and protocols. These include digital signatures, which provide a means of verifying the authenticity and integrity of digital messages or documents, and cryptographic hash functions, which generate fixed-length hash values from arbitrary input data. Hash functions are commonly used in password storage, digital certificates, and data integrity verification.

Key management is a critical aspect of cryptography, involving the generation, distribution, storage, and protection of cryptographic keys. Proper key management practices are essential for maintaining the security of cryptographic systems and preventing unauthorized access or misuse of keys.
Cryptography finds widespread application in numerous domains, including:

- Secure Communication: Encryption protocols like SSL/TLS (Secure Sockets Layer/Transport Layer Security) are used to secure communication over networks, such as web browsing, email, and instant messaging.
- Data Protection: Encryption is employed to protect sensitive data at rest (e.g., stored on disks or databases) and in transit (e.g., during file transfer or cloud storage).

- Authentication: Cryptographic techniques are used in authentication mechanisms, such as digital signatures and cryptographic authentication protocols, to verify the identity of users or entities.
- Access Control: Attribute-based encryption (ABE) and other cryptographic access control mechanisms enable fine-grained access control based on user attributes or roles.
- Blockchain Technology: Cryptography underpins the security and integrity of blockchain networks, ensuring the immutability of transaction data and the integrity of distributed ledger systems.

Overall, cryptography plays a crucial role in safeguarding sensitive information and ensuring the security and privacy of digital communications and transactions in an increasingly interconnected and data-driven world.

5.5 Sample Code

5.5.1 Upload.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Efficient;

import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;
import attributebased.Dbconnection;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.util.Arrays;
import java.util.Iterator;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger
```

```
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileItemFactory;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;

/**
 *
 * @author java2
 */
public class Upload extends HttpServlet {

    private static java.sql.Date getCurrentDate() {
        java.util.Date today = new java.util.Date();
        return new java.sql.Date(today.getTime());
    }

    /**
     * Processes requests for both HTTP <code>GET</code> and <code>POST</code>
     * methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
        protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            /* TODO output your page here. You may use following sample code. */
            Connection co;
            PreparedStatement pstm = null;
```

```
//      PreparedStatement pst2 = null;
        String filename = "";
         String address = "";
        String longi = "";
        String lati = "";
       String protocol = "";
       String quality1 = "";
       String email = (String) request.getSession().getAttribute("email");
       try {
          boolean isMultipartContent = ServletFileUpload.isMultipartContent(request);
          if (!isMultipartContent) {
             return;
          }
          FileItemFactory factory = new DiskFileItemFactory();
          ServletFileUpload upload = new ServletFileUpload(factory);
          try {
   List<FileItem> fields = upload.parseRequest(request);
   Iterator<FileItem> it = fields.iterator();
   if (!it.hasNext()) {
   return;
   }
   while (it.hasNext()) {
   FileItem fileItem = it.next();

//     if (fileItem.getFieldName().equals("quality1")) {
//    quality1 = fileItem.getString();
//    System.out.println("quality1" + quality1);
//    }

//    if (fileItem.getFieldName().equals("date")) {
//    date = fileItem.getString();
//    System.out.println("dt1" + date);
//    }
//



   if (fileItem.getFieldName().equals("address")) {
   address = fileItem.getString();
   System.out.println("File address" + address);
```

```
    if (fileItem.getFieldName().equals("longi")) {
    longi = fileItem.getString();
    System.out.println("File longi" + longi);
    }

    if (fileItem.getFieldName().equals("lati")) {
    lati = fileItem.getString();
    System.out.println("File lati" + lati);
    }
    if (fileItem.getFieldName().equals("protocol")) {
    protocol = fileItem.getString();
    System.out.println("File protocol" + protocol);
    }
if (fileItem.getFieldName().equals("filename")) {
    filename = fileItem.getString();
    System.out.println("File Name" + filename);
    }

    else {

    }
    boolean isFormField = fileItem.isFormField();
    if (isFormField) {
    } else {
    try {
    co = Dbconnection.getConnection();
                PreparedStatement pst2=co.prepareStatement("insert into upload
values(?,?,?,?,?,?,?,?,?,?)");

    System.out.println("getD " + fileItem.getName());
    String str = getStringFromInputStream(fileItem.getInputStream());

     KeyGenerator keyGen = KeyGenerator.getInstance("AES");
    keyGen.init(128);
    SecretKey secretKey = keyGen.generateKey();
    System.out.println("secret key:" + secretKey);

    byte[] be = secretKey.getEncoded();//encoding secretkey
     String skey = Base64.encode(be);
```

```
    System.out.println("converted secretkey to string:" + skey);
    String cipher = new encryption().encrypt(str, secretKey);
//    String date1 = date;
//    String q1 = quality1;
//     System.out.println("q1" + q1);



       pst2.setString(1,filename);
          pst2.setString(2,email);
       pst2.setString(3,str);
       pst2.setString(4,skey);
       pst2.setString(5,cipher);
        pst2.setString(6,address);
         pst2.setString(7,longi);
         pst2.setString(8,lati);
         pst2.setString(9,protocol);
         pst2.setString(10,"pending");



    int i = pst2.executeUpdate();
    if (i == 1) {
        response.sendRedirect("upload.jsp?m1=success");
    } else {
        response.sendRedirect("upload.jsp?msg1=failed");
    }


    co.close();
   } catch (Exception e) {
    out.println(e.toString());
   }
   }
   }
   } catch (FileUploadException e) {
   } catch (Exception ex) {
  Logger.getLogger(Upload.class.getName()).log(Level.SEVERE, null, ex);
```

```
        } finally {
            out.close();
        }
    }
    }

    private static String getStringFromInputStream(InputStream is) {
        BufferedReader br = null;
        StringBuilder sb = new StringBuilder();
        String line;
        try {
            br = new BufferedReader(new InputStreamReader(is));
            while ((line = br.readLine()) != null) {
                sb.append(line + "\n");
            }
        } catch (IOException e) {
        } finally {
            if (br != null) {
                try {
                    br.close();
                } catch (IOException e) {
                }
            }
        }
        return sb.toString();
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">

    /**
     * Handles the HTTP <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
        throws ServletException, IOException {
    processRequest(request, response);
}

/**
 * Handles the HTTP <code>POST</code> method.
 *
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}

/**
 * Returns a short description of the servlet.
 *
 * @return a String containing servlet description
 */
@Override
public String getServletInfo() {
    return "Short description";
}// </editor-fold>

private long[] splitToNChar(String str, int len) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
}

}
```

5.5.2 Encryption.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package Efficient;

import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;
import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.util.Scanner;

import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.JOptionPane;
import sun.misc.BASE64Encoder;

public class encryption {

   public String encrypt(String text, SecretKey secretkey) {
      String plainData = null, cipherText = null;
      try {
         plainData = text;

         Cipher aesCipher = Cipher.getInstance("AES");//getting AES instance
          aesCipher.init(Cipher.ENCRYPT_MODE, secretkey);//initiating ciper encryption
using secretkey

         byte[] byteDataToEncrypt = plainData.getBytes();
         byte[] byteCipherText = aesCipher.doFinal(byteDataToEncrypt);//encrypting data

               cipherText = new BASE64Encoder().encode(byteCipherText);//converting
encrypted data to string

             System.out.println("\n Given text : " + plainData + " \n Cipher Data : " +
cipherText);
```

```
        } catch (Exception e) {
            System.out.println(e);
        }
        return cipherText;
    }

}
```

5.5.3 Decryption.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package Efficient;

import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;
import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.util.Scanner;

import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.JOptionPane;
import sun.misc.BASE64Encoder;

public class encryption {

    public String encrypt(String text, SecretKey secretkey) {
        String plainData = null, cipherText = null;
        try {
            plainData = text;

            Cipher aesCipher = Cipher.getInstance("AES");//getting AES instance
```

```
aesCipher.init(Cipher.ENCRYPT_MODE, secretkey);//initiating ciper encryption
using secretkey

byte[] byteDataToEncrypt = plainData.getBytes();
byte[] byteCipherText = aesCipher.doFinal(byteDataToEncrypt);//encrypting data

cipherText = new BASE64Encoder().encode(byteCipherText);//converting
encrypted data to string

System.out.println("\n Given text : " + plainData + " \n Cipher Data : " +
cipherText);

} catch (Exception e) {
   System.out.println(e);
}
return cipherText; }}
```

5.5.4 UploadCloud.java

```
package attributebased;

import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.crypto.KeyGenerator;
```

```
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileItemFactory;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;

/**
 *
 * @author java2
 */
public class uploadcloud extends HttpServlet {

   //private static java.sql.Date getCurrentDate() //{
      //java.util.Date today = new java.util.Date();
      //return new java.sql.Date(today.getTime());
   //}

   /**
    * Processes requests for both HTTP <code>GET</code> and <code>POST</code>
    * methods.
    *
    * @param request servlet request
    * @param response servlet response
    * @throws ServletException if a servlet-specific error occurs
    * @throws IOException if an I/O error occurs
    */
       protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
           throws ServletException, IOException {
       response.setContentType("text/html;charset=UTF-8");
       try (PrintWriter out = response.getWriter()) {
          /* TODO output your page here. You may use following sample code. */
          Connection con = null;
          PreparedStatement pstm = null
```

```
 String filename = "";
 String owner = "";

 String privatekey = "";
 String policy = "";
 String time = "";
 String exp = "";
String branch = "";
String data = "";
String fname = "";
String t1 = "";

try {
   boolean isMultipartContent = ServletFileUpload.isMultipartContent(request);
   if (!isMultipartContent) {
      return;
   }
   FileItemFactory factory = new DiskFileItemFactory();
   ServletFileUpload upload = new ServletFileUpload(factory);
   try {
      List<FileItem> fields = upload.parseRequest(request);
      Iterator<FileItem> it = fields.iterator();
      if (!it.hasNext()) {
         return;
      }

      while (it.hasNext()) {
         FileItem fileItem = it.next();

          fname = fileItem.getName();
          if (fileItem.getFieldName().equals("privatekey")) {
            privatekey = fileItem.getString();
            System.out.println("File Name" + privatekey);

         }

            if (fileItem.getFieldName().equals("policy")) {
            policy = fileItem.getString();
            System.out.println("File Name" + policy);
```

```
}

    if (fileItem.getFieldName().equals("time")) {
   time = fileItem.getString();
   System.out.println("File Name" + time);

}

      if (fileItem.getFieldName().equals("exp")) {
   exp = fileItem.getString();
   System.out.println("File Name" + exp);

}

       if (fileItem.getFieldName().equals("branch")) {
   branch = fileItem.getString();
   System.out.println("File Name" + branch);

}

if (fileItem.getFieldName().equals("filename")) {
    filename = fileItem.getString();
    System.out.println("File Name" + filename);

}
if(fileItem.getFieldName().equals("owner")) {
  owner = fileItem.getString();
    System.out.println("File Keyword" + owner);


}
if(fileItem.getFieldName().equals("data")) {
  data = getStringFromInputStream(fileItem.getInputStream());
    System.out.println("data" + data);
}
if(fileItem.getFieldName().equals("t1")) {
  t1 = getStringFromInputStream(fileItem.getInputStream());
    System.out.println("t1" + t1);
```

```
}

            String polatt = "C:/Users/mahmo/OneDrive/Desktop/policy_attri.txt";
            String polatt1 = "C:/Users/mahmo/OneDrive/Desktop/keypolicy_attri.txt";
            String finalkey=null;

        // create list one and store values
         Map<String, String[]> map = new HashMap<String, String[]>();
          String[] attributesArray = new String[]{exp, branch};
        map.put(policy, attributesArray);



         FileWriter fstream;
         BufferedWriter output;
         fstream = new FileWriter(polatt);
         output = new BufferedWriter(fstream);

           for ( String key : map.keySet() )
           {

              finalkey = key;

            }

           for (int i=0;i<2;i++)
           {
                  output.write(finalkey  +" " + attributesArray[i]  );
            output.append(" ");
             }

            // code for writiing policy and respective key
            FileWriter fstream1;
        BufferedWriter output1;
        fstream1 = new FileWriter(polatt1);
        output1 = new BufferedWriter(fstream1);
        String key1= privatekey;
        String sales_value = policy;
```

```
output1.write(key1  +" " + sales_value +"\n");
output1.close();
output.close();


                try {
                   con = Dbconnection.getConnection();
                         pstm = con.prepareStatement("insert into uploadcloud (filename,
data, owner,skey,policy,time,exp,branch,t1)values(?,?,?,?,?,?,?,?,?,?)");

                   System.out.println(data);

                   pstm.setString(1, filename);
                   pstm.setString(2, data);
                   pstm.setString(3, owner);
                   pstm.setString(4, privatekey);
                   pstm.setString(5, policy);
                   pstm.setString(6, time);
                   pstm.setString(7, exp);
                   pstm.setString(8, branch);
                   pstm.setString(9, t1);

                   /*Cloud Start*/
                                                                File  f  =  new
File("C:\\Users\\mahmo\\OneDrive\\Desktop\\input"+fname);
                   FileWriter fw = new FileWriter(f);
                   fw.write(data);
                   fw.close();
                  Ftpcon ftpcon = new Ftpcon();
                   ftpcon.upload(f, filename);
                   /*Cloud End*/

                   int i = pstm.executeUpdate();

                     PreparedStatement ps=con.prepareStatement("update uploadcloud
set status='Uploaded' where filename='" + filename + "' ");
                    ps.executeUpdate();

                     PreparedStatement pst=con.prepareStatement("update upload set
status='Uploaded' where filename='" + filename + "' ");
```

```
                    pst.executeUpdate();

                  if (i == 1) {
                 response.sendRedirect("cpri.jsp?upload=File Upload Success");
                  } else {

                     response.sendRedirect("cpri.jsp?upload1=File Upload Failed");
                  }

                } catch (Exception e) {

                      PreparedStatement pst=con.prepareStatement("update upload set
status='Duplicate' where filename='" + filename + "' ");
                    pst.executeUpdate();
                   response.sendRedirect("cpri.jsp?upload2=File Already Uploaded");
                  out.println(e.toString());
                }


            } catch (FileUploadException e) {
            } catch (Exception ex) {
                  Logger.getLogger(uploadcloud.class.getName()).log(Level.SEVERE, null,
ex);
            }
         } finally {
            out.close();
         }
      }
   }

   private static String getStringFromInputStream(InputStream is) {
      BufferedReader br = null;
      StringBuilder sb = new StringBuilder();
      String line;
      try {
         br = new BufferedReader(new InputStreamReader(is));
         while ((line = br.readLine()) != null) {
            sb.append(line + "\n");
         }
      } catch (IOException e) {
```

```
        } finally {
            if (br != null) {
                try {
                    br.close();
                } catch (IOException e) {
                }
            }
        }
        return sb.toString();
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">

    /**
     * Handles the HTTP <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Handles the HTTP <code>POST</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
```

```
    }

    /**
     * Returns a short description of the servlet.
     *
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return "Short description";
    }

}
```

6. EXPERIMENT SCREENSHOTS

Fig 6.1 Owner Register and Login

Fig 6.2 End User register and Login

Fig 6.3 Central Attribute Authority

Fig 6.4 User File Request

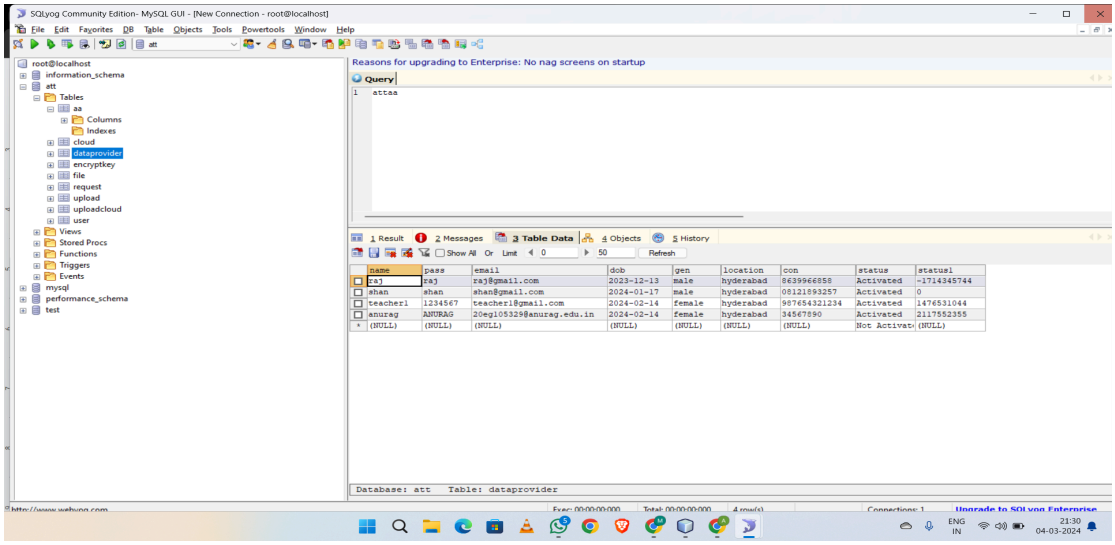


Fig 6.5 Data Owner Database

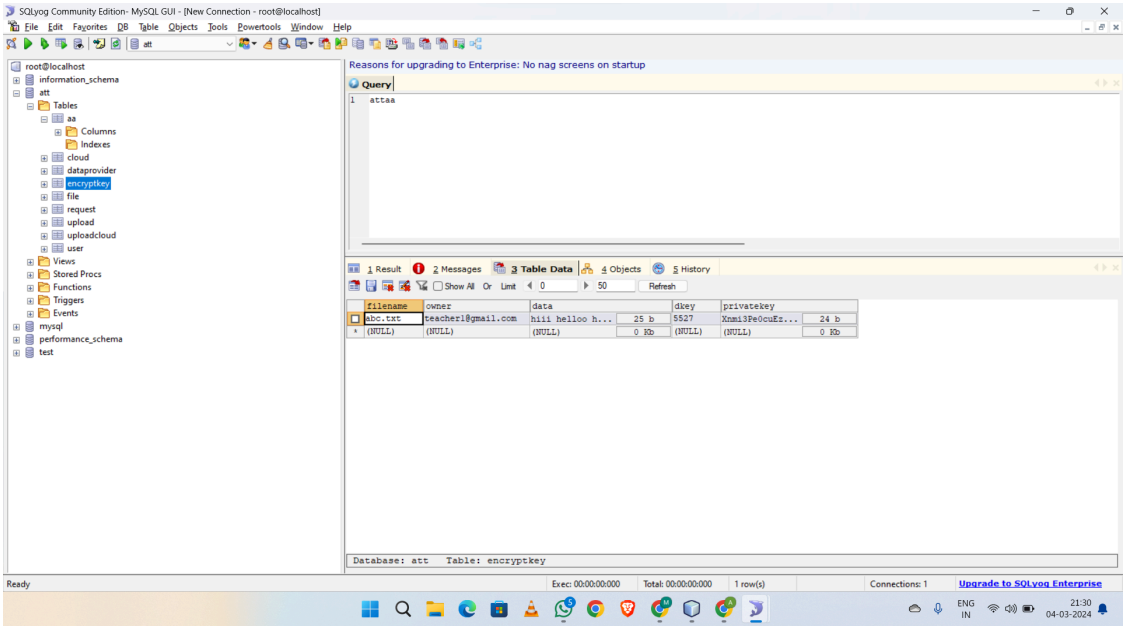


Fig 6.6 Secret Key Database

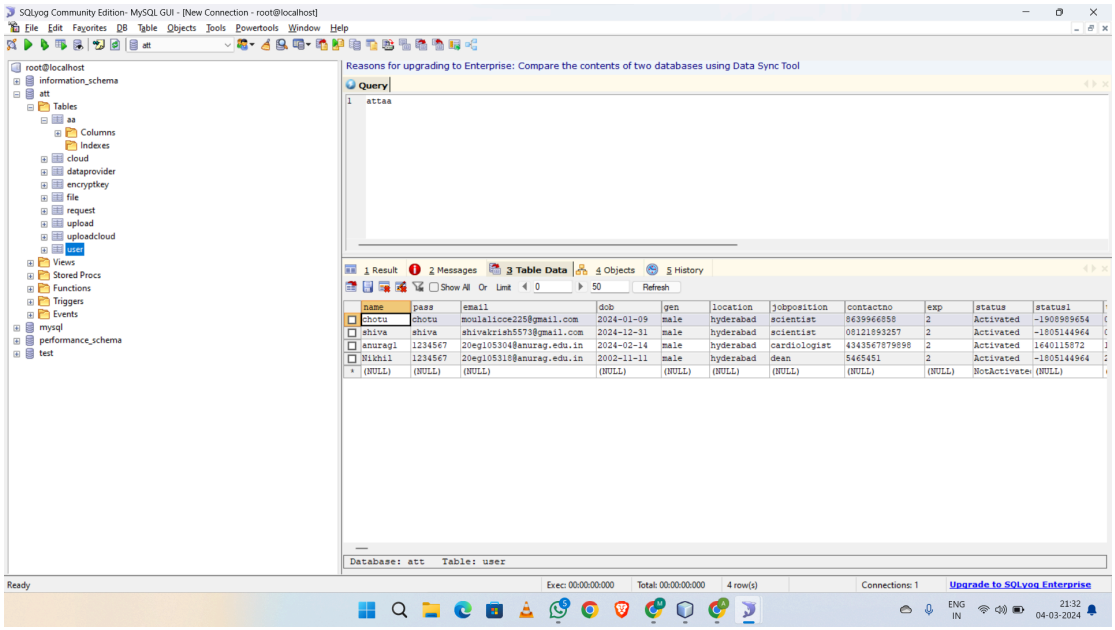


Fig 6.7 End User Database

7. EXPERIMENTAL RESULTS

7.1 Parameters and Formula

The proposed scheme Revocable Multi-Authority Attribute-Based Encryption with Time-Based Authority embraces five entities: central authority (CA), attribute authorities (AAs), cloud service provider (CSP), data owner (DO) and data consumer (DC)

- **CA Setup(λ)**→PP This algorithm is implemented by CA. It takes as input a security parameter λ, outputs public system parameters PP .
- **DC Reg(**$info_{DC}$ **)**→uid. CA carries out the algorithm, using the DC's information $info_{DC}$(e.g., name,birthday etc.) as input, and the identity uid as output.
- **AA Reg(**$info_{AA}$ **)**→aid. CA performs the algorithm, with AA's information $info_{AA}$ as input, and identity aid as output.

Data encryption:

- Encrypt((M, ρ),$\{APK_{aid_k}\}aid_k \in I_A$, m)→CT . DO carries out the algorithm by entering access structure(M, ρ), public keys $\{APK_{aid_k}\}aid_k \in I_A$ according to the set I_A related to (M, ρ)and the data m, to output ciphertext CT .

Data decryption:

- Decrypt(CT ,$\{SK_{uid,aid_k}\}_{aid_k \in I_A}$)→m. DC executes the algorithm by entering ciphertext CT,and the secret keys $\{SK_{uid,aid_k}\}_{aid_k \in I_A}$ related to the set I_A , to output the data m

Time attribute access control:

- A real-time attribute is an attribute whose values depend on time ti∈T. In our model, availability of a resource time (ti) environment for a subject determined based on the real-time attributes attr (x, ti),x∈ {s, o, e},with values in a linearly ordered set of availability labels L={`m=hi º`m−1º. . . º`1=lo}.2
- The availability label of attr(x, ti)is called priority when x is a subject, congestion when x is an object, and criticality when x is the environment. Availability labels are dynamically determined based on user events, the context of the requested service and system events.

Secret key generation:

- This phase is composed of the SKeyGen algorithm.
- SK eyGen(ASK_{aid} , $S_{uid,aid}$,{{$VK_{x_{aid}}$}$_{xaid \in S_{uid,aid}}$)→$SK_{uid,aid}$. AA performs the algorithm to generate secret key $SK_{uid,aid}$ for the DC.

Attribute revocation:

- This phase consists of three algorithms: UKeyGen,SKUpdate and CT Update.
- U KeyGen($ASK_{aid'}$,$\overline{x}_{aid'}$,$VK_{\overline{x}_{aid'}}$)→ $\overline{VK}_{\overline{x}_{aid'}}$, $UK_{\overline{x}_{aid'}}$. Outputs the new version key
- $\overline{VK}_{\overline{x}_{aid'}}$ and the update key $UK_{\overline{x}_{aid'}}$.
- SK Update($SK_{uid,aid'}$, $UK_{\overline{x}_{aid'}}$)→$\overline{SK}_{uid,aid'}$. Output is the new secret key $\overline{SK}_{uid,aid'}$.
- CT Update(CT, $UK_{\overline{x}_{aid'}}$)→$\overline{CT}$. output is the new ciphertext $\overline{CT}$.

8. DISCUSSION OF RESULTS

The implementation of Revocable Multi-Authority Attribute-Based Encryption with Time-Based Authority yields significant advancements in secure data sharing and access control. This cryptographic scheme provides heightened security by encrypting data based on attributes, allowing for fine-grained access control enforced dynamically over time. Resulting from its revocable nature, administrators can efficiently manage access rights, adjusting them as per evolving security needs or user roles. Moreover, the scheme's multi-authority structure ensures scalability, accommodating systems with multiple governing bodies while preserving privacy by granting access based on attributes rather than revealing user identities. Overall, our method with Time-Based Authority offers an effective solution for secure and flexible information sharing across diverse applications, including cloud computing and healthcare systems.

8. a Experiment 1

Encryption Time Vs File Size(KB)

This implies that as the file size increases, the encryption time also increases linearly.

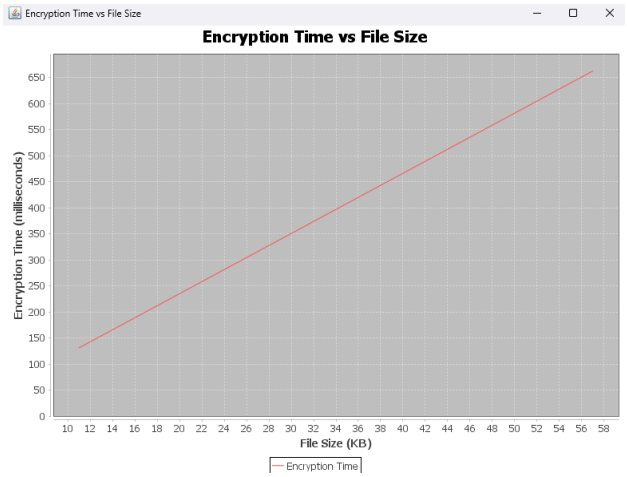


Fig 8.1 Encryption Time Vs File Size(KB) Graph

8. b Experiment 2:

Decryption Time Vs File Size(KB)

The decryption time may exhibit a linear relationship with file size.The graph illustrate how decryption time changes as file size increases.

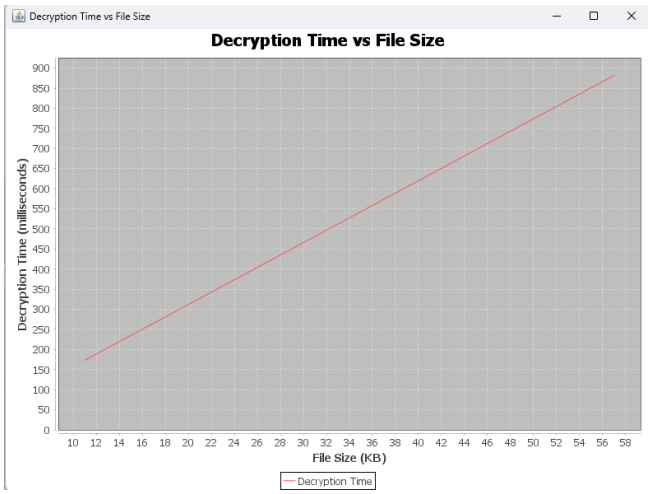


Fig 8.2 Decryption Time Vs File Size(KB) Graph

8. c Experiment 3:

CipherText Update Time Vs Number of Revoked Attributes

The graph will demonstrate how ciphertext update time varies with the number of revoked attributes.

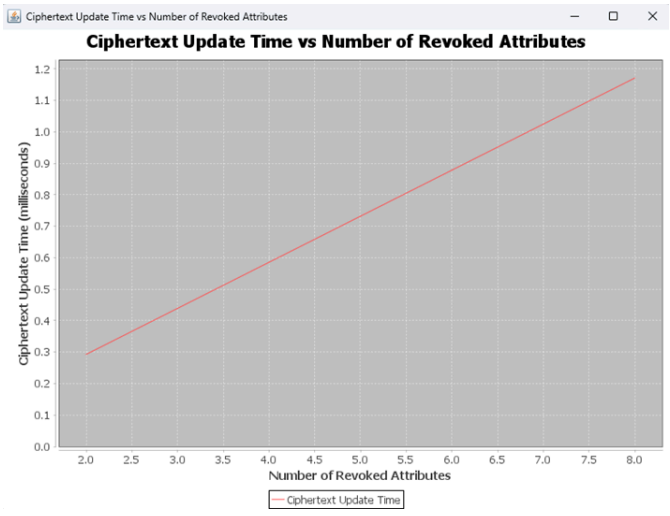


Fig 8.3 CipherText Update Time Vs Number of Revoked Attributes Graph

9. CONCLUSION

In conclusion, the development of Revocable Multi-Authority Attribute-Based Encryption (RMA-ABE) for time-based authorities represents a significant advancement in secure data access control systems. By introducing the capability to revoke access based on time-sensitive attributes, this scheme enhances the flexibility and granularity of access control policies. With the ability to dynamically adjust access privileges over time, organizations can better manage evolving user roles and permissions while maintaining data security. This innovation addresses the practical need for efficient and scalable access control mechanisms in dynamic environments. Furthermore, the integration of multiple authorities enables distributed attribute management, enhancing scalability and decentralization. Overall, RMA-ABE for time-based authorities offers a robust solution for fine-grained access control in modern data management systems, balancing security requirements with the need for flexibility and efficiency.

10. FUTURE ENHANCEMENTS

Enhancements for Revocable Multi-Authority Attribute-Based Encryption with Time-Based Authority (RMA-ABE-TBA) could focus on addressing existing limitations, improving performance, and enhancing security. Here are some potential future enhancements:

- **Efficiency Improvements:**
  - Develop more efficient algorithms for key generation, encryption, and decryption to reduce computational overhead.
  - Investigate techniques to optimize attribute revocation processes without compromising security.
  - Explore parallelization and distributed computing approaches to enhance scalability and performance.
- **Enhanced Security:**
  - Conduct rigorous security analysis to identify and mitigate potential vulnerabilities, such as side-channel attacks or collusion attacks.
  - Research post-quantum cryptographic techniques to ensure resilience against quantum computing threats.
  - Integrate advanced cryptographic primitives for stronger security guarantees, such as homomorphic encryption or zero-knowledge proofs.
- **Fine-Grained Access Control:**
  - Extend the attribute-based access control model to support more expressive policies, including hierarchical or conditional attribute-based access control.
  - Investigate dynamic policy updates to enable real-time adaptation of access control policies based on changing requirements or contexts.
- **Usability and Interoperability:**
  - Develop user-friendly interfaces and tools to simplify the configuration, management, and monitoring of attribute-based encryption systems.
  - Enhance interoperability with existing cryptographic standards and protocols to facilitate integration with other systems and applications.
- **Scalability and Flexibility:**
  - Explore distributed attribute authority models to distribute the management of attributes and policies across multiple entities, enhancing scalability and fault tolerance.
  - Investigate techniques for efficiently handling large-scale attribute-based encryption systems with a high volume of users and data.

- **Privacy-Preserving Enhancements:**
    - Research techniques to enhance privacy-preserving capabilities, such as differential privacy mechanisms or secure multi-party computation, to protect sensitive attributes and access patterns.
    - Explore decentralized identity management approaches to minimize reliance on centralized authorities and enhance user privacy.
- **Standardization and Adoption:**
    - Contribute to standardization efforts within relevant industry or standardization bodies to establish interoperable frameworks and protocols for attribute-based encryption.
    - Promote awareness and adoption of attribute-based encryption techniques through education, training, and industry collaboration initiatives.

By focusing on these areas, future enhancements can strengthen the security, efficiency, usability, and scalability of Revocable Multi-Authority Attribute-Based Encryption with Time-Based Authority, making it more suitable for a wide range of applications and deployment scenarios.

11. REFERENCES

[1] Revocable Attribute-Based Encryption With Data Integrity in Clouds. Chunpeng Ge; Willy Susilo; Joonsang Baek; Zhe Liu; Jinyue Xia; Liming Fang. IEEE(2021)

[2] Vipul Goyal,Omkant Pandey, Amit Sahai, Brent Waters, "Attribute-based encryption for fine-grained access control of encrypted data",Oct.2006.

[3] K. Sethi, A. Pradhan, and P. Bera, "Practical traceable multi-authority CP-ABE with outsourcing decryption and access policy updation," Journal ofInformation Security and Applications, vol. 51, pp. 102435-102450, Apr.2020

[4] Q. Xu, C. Tan, W. Zhu, Y. Xiao, Z. Fan, and F. Cheng, "Decentralized Attribute-based conjunctive keyword search scheme with online/offlineencryption and outsource decryption for cloud computing," Future Gen-eration Computer Systems, vol. 97, pp. 306-326, Mar. 2019

[5] S.Ding, C. Li, and H. Li, "A novel efficient pairing-free CP-ABE based on elliptic curve cryptography for IoT," IEEE Access, vol. 6, pp. 27336- 27345, May. 2018

[6] J. Li, X. Lin, Y. Zhang, and J. Han, "KSF-OABE: Outsourced attribute-based encryption with keyword search function for cloud storage," IEEET. Ser. Comput., vol. 10, no. 5, pp. 715-725, Dec. 2017

[7] Z. Liu, Z. Jiang, and X. Wang, "Practical attribute-based encryption:Outsourcing decryption, attribute revocation and policy updating," Journal Of Network and Computer Applications, vol. 108, pp. 112-123, 2018