

SECURE CLOUD STORAGE : MULTI AUTHORITY ACCESS AND ENCRYPTION

*Major project report submitted
in partial fulfillment of the requirement for award of the degree of*

**Bachelor of Technology
in
Computer Science & Engineering**

By

K SHIVA SHANKAR	(20UECS0495)	(VTU16863)
A HARI VARA PRASAD	(20UECS0003)	(VTU17077)
M NIRANJAN REDDY	(20UECS0636)	(VTU16855)

*Under the guidance of
Dr. M. Sankar, M.Tech., Ph.D.,
Professor*



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING**

**VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF
SCIENCE & TECHNOLOGY**

(Deemed to be University Estd u/s 3 of UGC Act, 1956)

**Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA**

May, 2024

SECURE CLOUD STORAGE : MULTI AUTHORITY ACCESS AND ENCRYPTION

*Major project report submitted
in partial fulfillment of the requirement for award of the degree of*

**Bachelor of Technology
in
Computer Science & Engineering**

By

K SHIVA SHANKAR (20UECS0495) **(VTU16863)**
A HARI VARA PRASAD (20UECS0003) **(VTU17077)**
M NIRANJAN REDDY (20UECS0636) **(VTU16855)**

*Under the guidance of
Dr. M. Sankar, M.Tech., Ph.D.,
Professor*



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING**

**VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF
SCIENCE & TECHNOLOGY**

(Deemed to be University Estd u/s 3 of UGC Act, 1956)

**Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA**

May, 2024

CERTIFICATE

It is certified that the work contained in the project report titled “SECURE CLOUD STORAGE : MULTI AUTHORITY ACCESS AND ENCRYPTION” by A HARI VARA PRASAD (20UECS0003), M NIRANJAN REDDY (20UECS0636), K SHIVA SHANKAR (20UECS0495) has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Signature of Supervisor

Computer Science & Engineering

School of Computing

Vel Tech Rangarajan Dr. Sagunthala R&D

Institute of Science & Technology

May, 2024

Signature of Professor In-charge

Computer Science & Engineering

School of Computing

Vel Tech Rangarajan Dr. Sagunthala R&D

Institute of Science & Technology

May, 2024

DECLARATION

We declare that this written submission represents our ideas in our own words and where others ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

A HARI VARA PRASAD

Date: / /

M NIRANJAN REDDY

Date: / /

K SHIVA SHANKAR

Date: / /

APPROVAL SHEET

This project report entitled “SECURE CLOUD STORAGE - MULTI AUTHORITY ACCESS AND ENCRYPTION” by M NIRANJAN REDDY (20UECS0636), K SHIVA SHANKAR (20UECS0495), A HARI VARA PRASAD (20UECS0003) is approved for the degree of B.Tech in Computer Science & Engineering.

Examiners

Supervisor

Dr. M Sankar, M.Tech., Pd.D.,

Date: / /

Place:

ACKNOWLEDGEMENT

We express our deepest gratitude to our respected **Founder Chancellor and President Col. Prof. Dr. R. RANGARAJAN B.E. (EEE), B.E. (MECH), M.S (AUTO),D.Sc., Foundress President Dr. R. SAGUNTHALA RANGARAJAN M.B.B.S.** Chairperson Managing Trustee and Vice President.

We are very much grateful to our beloved **Vice Chancellor Prof. S. SALIVAHANAN**, for providing us with an environment to complete our project successfully.

We record indebtedness to our **Professor & Dean, Department of Computer Science & Engineering, School of Computing, Dr. V. SRINIVASA RAO, M.Tech., Ph.D.**, for immense care and encouragement towards us throughout the course of this project.

We are thankful to our **Head, Department of Computer Science & Engineering, Dr. M. S. MURALI DHAR, M.E., Ph.D.**, for providing immense support in all our endeavors.

We also take this opportunity to express a deep sense of gratitude to our Internal Supervisor **Dr. M. SANKAR, M.Tech., Ph.D.**, for his cordial support, valuable information and guidance, he helped us in completing this project through various stages.

A special thanks to our **Project Coordinators Mr. V. ASHOK KUMAR, M.Tech., Ms. C. SHYAMALA KUMARI, M.E.**, for their valuable guidance and support throughout the course of the project.

We thank our department faculty, supporting staff and friends for their help and guidance to complete this project.

K SHIVA SHANKAR	(20UECS0495)
M NIRANJAN REDDY	(20UECS0636)
A HARI VARA PRASAD	(20UECS0003)

ABSTRACT

The integration of a multi-authority authentication system with a file encryption mechanism is a critical development in enhancing security in cloud storage environments. The approach addresses the challenges of securing data in cloud storage by combining the authentication capabilities of multiple authorities with robust encryption techniques. By utilizing a multi-authority system, the authentication process becomes more resilient to potential breaches or insider threats, as it requires authorization from multiple sources. This added layer of security significantly reduces the risk of unauthorized access to sensitive data stored in the cloud. Furthermore, the integration with a file encryption mechanism ensures that even if unauthorized access is gained, the encrypted data remains incomprehensible and protected. This comprehensive approach to security in cloud storage environments not only safeguards data from unauthorized access but also provides a robust defense against cyberattacks and data breaches. Overall, the integration of a multi-authority authentication system with file encryption mechanisms offers a sophisticated and effective solution to enhance security and privacy in cloud storage environments.

Keywords: Authentication resilience, Cloud storage security, Cybersecurity, Data breaches, File encryption mechanism, Insider threats, Multi-authority authentication, Robust encryption techniques, Sensitive data protection, Unauthorized access.

LIST OF FIGURES

4.1	Architecture Diagram	15
4.2	Data flow diagram	16
4.3	Use case diagram	17
4.4	Class diagram	18
4.5	Sequence diagram	19
4.6	Activity diagram	20
5.1	File Selection and upload Window	27
5.2	File Encrypted and uploaded to Cloud	30
5.3	Successful connection of Database	38
5.4	GUI Home Window	40
6.1	Multi Level Authentication using Keys	46
8.1	Plagiarism report	50
9.1	Poster	55

LIST OF ACRONYMS AND ABBREVIATIONS

AES	Advanced Encryption Standard
CSE	Cloud Security Enhancement
DTD	Data Transfer and Decryption
ESM	Encryption Mechanism
EAS	Enhanced Authentication System
ESM	Enhanced Security Mechanism
FM	File Management
FS	File System
IT	Integration Technology
MAS	Multi-Authority System
MAAS	Multi-Authority Authentication System
MFA	Multi-Factor Authentication
SEC	Security
TFA	Two-Factor Authentication

TABLE OF CONTENTS

	Page.No
ABSTRACT	v
LIST OF FIGURES	vi
LIST OF ACRONYMS AND ABBREVIATIONS	vii
1 INTRODUCTION	1
1.1 Introduction	1
1.2 Aim of the Project	1
1.3 Project Domain	2
1.4 Scope of the Project	3
2 LITERATURE REVIEW	4
3 PROJECT DESCRIPTION	7
3.1 Existing System	7
3.2 Proposed System	9
3.3 Feasibility Study	9
3.3.1 Technical Feasibility	10
3.3.2 Social Feasibility	11
3.4 System Specification	12
3.4.1 Hardware Specification	12
3.4.2 Software Specification	12
3.4.3 Standards and Policies	14
4 METHODOLOGY	15
4.1 General Architecture	15
4.2 Design Phase	16
4.2.1 Data Flow Diagram	16
4.2.2 Use Case Diagram	17
4.2.3 Class Diagram	18
4.2.4 Sequence Diagram	19

4.2.5	Activity Diagram	20
4.3	Algorithm & Pseudo Code	22
4.3.1	Algorithm Used: Advanced Encryption Standard (AES)	22
4.3.2	Pseudo Code	23
4.4	Steps to execute/run/implement the project	25
4.4.1	Step 1: Requirement Analysis:	25
4.4.2	System Design:	25
4.4.3	Select Technologies:	25
4.4.4	Implement Multi-Authority Authentication:	25
4.4.5	File Encryption Implementation:	25
4.4.6	Integration with Cloud Storage:	25
4.4.7	Testing:	26
4.4.8	Security Review:	26
4.4.9	Deployment:	26
4.4.10	Monitoring and Maintenance:	26
5	IMPLEMENTATION AND TESTING	27
5.1	Input and Output	27
5.1.1	Input Design	27
5.1.2	Implement Multi-Authority Authentication:	27
5.1.3	Output Design	30
5.2	Testing	31
5.3	Types of Testing	31
5.3.1	Unit Testing	31
5.3.2	Integration Testing	35
5.3.3	System Testing	39
5.3.4	Test Result	40
6	RESULTS AND DISCUSSIONS	41
6.1	Efficiency of the Proposed System	41
6.2	Comparison of Existing and Proposed System	42
6.3	Sample Code	43
7	CONCLUSION AND FUTURE ENHANCEMENTS	48
7.1	Conclusion	48
7.2	Future Enhancements	48

8	PLAGIARISM REPORT	50
9	SOURCE CODE & POSTER PRESENTATION	51
9.1	Source Code	51
9.2	Poster Presentation	55
	References	56

Chapter 1

INTRODUCTION

1.1 Introduction

Multi-Authority Authentication Systems have emerged as a crucial component in enhancing the security of cloud storage environments by enabling secure access control mechanisms. These systems are designed to address the limitations of traditional single-authority authentication models by distributing the authentication authority across multiple entities, thus reducing the risk of a single point of failure. The integration of a Multi-Authority Authentication System with a file encryption mechanism further strengthens security measures by ensuring that only authorized users can access and decrypt sensitive data stored in the cloud. This integration enhances data confidentiality and integrity, as well as providing a robust defense against unauthorized access and data breaches. By leveraging multiple authorities for authentication and encryption, this approach offers a higher level of security and trust in cloud storage environments, making it an ideal solution for organizations seeking to safeguard their sensitive information from potential threats and vulnerabilities. File Encryption Mechanisms in Cloud Storage play a crucial role in ensuring data security and privacy within cloud storage environments. One common mechanism is symmetric key encryption, where a single key is used for both encryption and decryption of files. This method is efficient and fast, making it suitable for large-scale data storage in the cloud. Another encryption mechanism is public key encryption, which utilizes a pair of keys - a public key for encryption and a private key for decryption.

1.2 Aim of the Project

The project aims to bolster security in cloud storage environments by combining a multi-authority authentication system with robust file encryption techniques. This integration seeks to address the challenges posed by potential breaches and insider threats, ensuring a resilient authentication process. By leveraging multiple authorities, the system enhances authentication resilience, requiring authorization from var-

ious sources to mitigate unauthorized access risks. Furthermore, the incorporation of a file encryption mechanism safeguards sensitive data, even in the event of unauthorized access. Ultimately, the project endeavors to provide a comprehensive solution that not only protects data from breaches but also fortifies against cyberattacks, enhancing security and privacy in cloud storage environments.

1.3 Project Domain

The integration of a multi-authority authentication system with a file encryption mechanism represents a critical advancement in the domain of cloud storage security. In today's digital landscape, where vast amounts of data are stored and accessed remotely, ensuring the confidentiality and integrity of this data is paramount. Traditional singleauthority authentication systems are susceptible to various security risks, including insider threats and brute force attacks. By transitioning to a multi-authority authentication model, the project aims to mitigate these risks by requiring authorization from multiple independent sources. This approach enhances authentication resilience, making it significantly more difficult for malicious actors to compromise user credentials and gain unauthorized access to sensitive data.

Furthermore, the integration with a file encryption mechanism adds an additional layer of security to the cloud storage environment. Even if unauthorized access is somehow obtained, the encrypted data remains incomprehensible and protected. This ensures that even in the event of a breach, the potential impact is minimized, as the encrypted data is rendered useless to unauthorized parties. By combining multi-authority authentication with robust file encryption techniques, the project aims to provide a comprehensive solution to enhance security in cloud storage environments. This approach not only safeguards data from unauthorized access but also strengthens defenses against cyberattacks and data breaches, thereby bolstering trust and confidence in cloud-based storage solutions.

1.4 Scope of the Project

The scope of the project encompasses the design, implementation, and evaluation of a comprehensive security framework for cloud storage environments. The primary focus lies in integrating a multi-authority authentication system with a file encryption mechanism to bolster data protection and access control. This involves conducting an indepth analysis of existing authentication and encryption protocols to identify suitable methods for integration. The project will also involve developing software modules to facilitate seamless interaction between the authentication system and the encryption mechanism. Furthermore, extensive testing will be conducted to assess the performance, scalability, and resilience of the integrated solution under various scenarios, including simulated cyberattacks and unauthorized access attempts.

In addition to technical implementation, the scope extends to addressing usability and scalability considerations to ensure the practical applicability of the solution in real-world cloud storage environments. This entails exploring user authentication workflows, user experience enhancements, and scalability measures to accommodate growing storage demands and user bases. Moreover, the project will delve into regulatory compliance requirements and privacy considerations to ensure adherence to relevant standards and regulations governing data protection in cloud environments. By encompassing these aspects, the project aims to deliver a robust, scalable, and compliant security solution that effectively mitigates risks associated with unauthorized access and data breaches in cloud storage environments.

Chapter 2

LITERATURE REVIEW

[1] P. S. Challagidad, M. N. Birje et al. (2020) [10] highlighted the significance of efficient multi-authority access control using attribute-based encryption (ABE) in cloud storage. Their study emphasizes the need for enhanced security measures in cloud environments and proposes a system that allows fine-grained access control based on user attributes. By leveraging ABE, the system ensures that only authorized users with specific attributes can access sensitive data, thereby addressing concerns related to data privacy and security in cloud storage.

[2] J. Gu, J. Shen, B. Wang et al. (2021) [16] explored the development of a robust and secure multi-authority access control system tailored for cloud storage. Their research focuses on integrating advanced authentication and encryption mechanisms to safeguard data integrity and confidentiality. By enhancing scalability and resilience through the incorporation of multiple authorities, their system offers a comprehensive solution for secure data management in cloud environments.

[3] R. Gupta et al. (2023) [5] introduced a secured and privacy-preserving multi-authority access control system designed for cloud-based healthcare data sharing. Recognizing the sensitive nature of healthcare information, their system employs cutting-edge encryption techniques to ensure data security and confidentiality. By prioritizing privacy preservation, their research addresses the unique challenges associated with sharing healthcare data in cloud environments, laying the foundation for secure and compliant data sharing practices in the healthcare industry.

[4] N. S. Havanje et al. (2022) [29] proposed a secure and reliable data access control mechanism tailored for multicloud environments. Their research addresses vulnerabilities in inter-server communication, a critical aspect of cloud security often overlooked in traditional access control systems. By enhancing communication security between cloud servers, their mechanism ensures the integrity and confidentiality of data across multiple cloud platforms, contributing to overall cloud storage

security.

[5] M. B. Gunjal, V. R. Sonawane et al. (2023) [18] introduced a multi-authority access control mechanism focusing on role-based access control (RBAC) for data security in the cloud environment. Their system offers granular control over user privileges based on predefined roles, enhancing data security measures in cloud storage. By implementing RBAC principles, their research aims to restrict access to sensitive information to authorized users only, addressing the growing demand for robust access control mechanisms in cloud computing.

[6] J. S. Jayaprakash et al. (2020) [22] presented a novel approach to cloud data encryption and authentication using an enhanced Merkle hash tree method. By leveraging Merkle hash trees, their system facilitates secure data transmission and authentication, mitigating the risk of unauthorized access and tampering. Their research contributes to strengthening the security posture of cloud storage systems, particularly in scenarios where data integrity and authenticity are paramount.

[7] A. Kumar et al. (2023) [14] addressed privacy concerns in healthcare data management by proposing a multi-authority access control mechanism with anonymous authentication for maintaining personal health records (PHRs). Their system offers a comprehensive solution that protects patient privacy while ensuring authorized access to healthcare data. By incorporating anonymous authentication, their research aims to mitigate the risk of identity exposure and unauthorized data access in PHR systems, fostering trust and compliance in healthcare data sharing practices.

[8] A. Abirami et al. (2023) [11] focused on enhancing cloud storage security through the use of multi-authority privacy-preserving data signcryption. By leveraging Rijndael encryption and HMAC, their system offers robust protection against data breaches and unauthorized access. Through signcryption, their research achieves a balance between privacy preservation and data integrity, ensuring that sensitive information remains confidential and tamper-proof. Their study addresses the evolving security challenges in cloud storage, offering a promising solution for safeguarding data in cloud environments.

[9] K. Dhal et al. (2021) [25] proposed CEMAR, a fine-grained access control

system with a revocation mechanism for centralized multi-authority cloud storage. Their research enhances security and control over data access in cloud storage environments, addressing the need for granular access control mechanisms. By incorporating a revocation mechanism, their system enables efficient management of user permissions and enhances overall cloud storage security.

[10] S. Xiong et al. (2021) [17] introduced SEM-ACSIT, a secure and efficient multi-authority access control system for IoT cloud storage. Their research aims to strengthen security measures for Internet of Things (IoT) data in cloud environments. By offering a secure and efficient access control system, their study contributes to ensuring the integrity and confidentiality of IoT data stored in the cloud, addressing concerns related to data privacy and security in IoT deployments.

Chapter 3

PROJECT DESCRIPTION

3.1 Existing System

The existing system for integrating a multi-authority authentication system with a file encryption mechanism in cloud storage environments aims to address inherent vulnerabilities in traditional single-authority authentication and encryption methods. Single-authority authentication, prevalent in many cloud storage systems, poses risks of security breaches and unauthorized access if the authentication server is compromised. Similarly, conventional file encryption mechanisms may lack the robustness needed to counter various security threats effectively. In response, the proposed integration employs multiple authentication authorities to distribute the authentication process, reducing the vulnerability of a single point of failure. This multi-authority approach enhances security by necessitating authentication from diverse sources, thus lessening the impact of compromised authentication servers. Moreover, coupling this authentication system with robust file encryption strengthens overall security by employing advanced encryption algorithms and key management practices to safeguard sensitive data from unauthorized access and breaches.

However, despite its security advantages, the integrated system faces several challenges. Managing multiple authorities can lead to complexities in ensuring seamless access for authorized users, potentially resulting in authentication errors and access issues. Additionally, the decentralized nature of the system may pose governance challenges, including inconsistencies in policy enforcement and difficulties in monitoring and auditing user activities across different authorities. Furthermore, reliance on file encryption may introduce performance overhead, especially in large-scale cloud storage environments, impacting access speed and system efficiency. Interoperability issues with existing cloud storage platforms may also hinder adoption and integration. Addressing these challenges effectively is crucial to ensuring the overall effectiveness and usability of the integrated system in real-world cloud storage environments.

Advantages:

- **Increased Trust:** The integration of a multi-authority authentication system instills greater confidence in users regarding the verification of their identities. With multiple authorities involved, there's a higher assurance level in the authentication process, enhancing trust in the security of the system.
- **Reduced Risk of Unauthorized Access:** By requiring authorization from multiple sources, the system significantly reduces the risk of unauthorized access to sensitive data stored in the cloud. This helps in safeguarding confidential information from potential breaches or insider threats.
- **Enhanced Data Security:** The integration of a file encryption mechanism ensures that data remains protected, even if unauthorized access is gained. Encrypting files before uploading them to the cloud adds an extra layer of security, making it difficult for unauthorized users to interpret the data, thus enhancing overall data security.

Disadvantages:

- **Complexity in Management:** Managing multiple authentication authorities can lead to difficulties in ensuring seamless access for all authorized users. This complexity may result in authentication errors and access issues, potentially impacting user experience and system usability.
- **Governance Challenges:** The decentralized nature of the multi-authority system may introduce governance challenges, such as inconsistencies in policy enforcement and difficulties in monitoring and auditing user activities across different authorities. This can hinder effective governance and compliance with regulatory requirements.
- **Performance Overhead:** Reliance on file encryption for security can introduce performance overhead, particularly in large-scale cloud storage environments. This overhead may impact access speed and overall system efficiency, affecting user productivity and satisfaction.
- **Interoperability Issues:** The existing system may face interoperability challenges with existing cloud storage. These challenges could limit the system's adoption and integration capabilities, hindering its effectiveness in diverse cloud environments.

3.2 Proposed System

The proposed system aims to enhance security in cloud storage environments through the integration of a multi-authority authentication system and a file encryption mechanism. By employing a multi-authority authentication system, users' trust in identity verification processes will increase, as multiple authorities collectively validate user credentials, thereby reducing the risk of unauthorized access to sensitive data stored in the cloud. Additionally, the integration of a file encryption mechanism adds an extra layer of security by encrypting files before uploading them to the cloud, ensuring that even if unauthorized users gain access to the storage, the encrypted files remain protected. Overall, this system provides users with a secure platform to store their data, mitigating the risks of data breaches and cyberattacks, and offering peace of mind regarding data security.

The integration of a multi-authority authentication system with a file encryption mechanism offers significant advantages for enhancing security in cloud storage environments. This system enables the collaboration of multiple authorities in the authentication process, enhancing the reliability of user identity verification and reducing the risk of unauthorized data access. Moreover, the incorporation of file encryption ensures data security both in transit and at rest, adding an additional layer of protection against potential security breaches. By addressing both authentication and data protection aspects, the proposed system offers a comprehensive security solution that instills confidence in users regarding the safety and privacy of their stored information, making it invaluable for organizations and individuals seeking to bolster their cloud security measures.

3.3 Feasibility Study

A feasibility study is essential for determining the viability of integrating a multi-authority authentication system with a file encryption mechanism for enhanced security in cloud storage environments. This study would involve assessing the technical, operational, economic, and legal aspects of such integration. From a technical perspective, it is crucial to evaluate the compatibility of the authentication system and encryption mechanism with existing cloud storage infrastructures. This includes considering factors such as scalability, performance impact, and interoperability with

different cloud service providers. Operationally, the feasibility study should analyze how the integration would impact user experience, system management, and overall efficiency of data access and sharing processes. From an economic standpoint, cost-benefit analysis would be necessary to determine the financial implications of implementing and maintaining the integrated system in comparison to the potential security benefits it offers. Additionally, legal considerations such as data privacy regulations and compliance requirements must be assessed to ensure that the integration does not pose any legal risks or liabilities. By conducting a thorough feasibility study encompassing these various factors, stakeholders can make informed decisions regarding the implementation of a multi-authority authentication system with a file encryption mechanism for enhanced security in cloud storage environments.

3.3.1 Technical Feasibility

Technical feasibility is an essential aspect when considering the integration of a multiauthority authentication system with a file encryption mechanism for enhanced security in cloud storage environments. This feasibility study aims to assess the practicality and viability of implementing such a system from a technological perspective. The integration of a multi-authority authentication system with file encryption in cloud storage environments involves various technical considerations. One primary aspect is the compatibility of the authentication system with existing cloud storage platforms. It is crucial to ensure that the authentication system can seamlessly integrate with popular cloud services such as Amazon Web Services, Microsoft Azure, or Google Cloud Platform. Another important technical consideration is the encryption mechanism. The file encryption must be robust and secure to protect data from unauthorized access or tampering. This involves selecting appropriate encryption algorithms, key management strategies, and data protection protocols to ensure the confidentiality and integrity of stored information. Furthermore, the scalability of the system is a critical factor to consider. As cloud storage environments often handle large volumes of data and user requests, the authentication system and encryption mechanism must be able to scale efficiently to accommodate increasing demand without compromising performance or security. Additionally, the implementation of a multi-authority authentication system requires advanced technical expertise in the areas of identity management, access control, and secure communication protocols. It is essential to have skilled professionals who can design, develop, and maintain the

system effectively. Overall, the technical feasibility of integrating a multi-authority authentication system with a file encryption mechanism for enhanced security in cloud storage environments depends on various factors such as compatibility, security, scalability, and technical expertise. A thorough assessment of these aspects is necessary to ensure the successful implementation and operation of the system.

3.3.2 Social Feasibility

Social feasibility is an essential aspect to consider when integrating a multi-authority authentication system with a file encryption mechanism for enhanced security in cloud storage environments. This concept refers to the examination of how well a proposed system aligns with the social norms, preferences, and capabilities of the individuals who will be using it. In terms of the integration of a multi-authority authentication system and file encryption mechanism, social feasibility involves assessing whether users will be willing and able to adopt and effectively utilize the system. This includes considerations such as user acceptance, ease of use, user training requirements, and potential cultural or organizational barriers to implementation. User acceptance is a key factor in social feasibility. It is important to determine whether the proposed system meets the needs and expectations of the users. Users may be more likely to adopt the system if it offers clear benefits, such as improved security and ease of access to cloud storage resources. Ease of use is another critical aspect of social feasibility. The system should be intuitive and userfriendly, requiring minimal training for users to understand how to interact with it effectively. If the system is overly complex or difficult to use, it may be met with resistance from users. User training requirements should also be considered in the context of social feasibility. If extensive training is needed to use the system effectively, this could pose a barrier to adoption. Providing adequate training and support to users can help increase their comfort and confidence in using the system. Finally, cultural and organizational factors should be taken into account when assessing social feasibility. Different cultural norms and organizational structures may influence how users interact with technology and affect their willingness to adopt new systems.

These factors should be carefully considered and addressed to ensure successful implementation of the integrated authentication and encryption system. In conclu-

sion, social feasibility is a crucial consideration when integrating a multi-authority authentication system with a file encryption mechanism in cloud storage environments. By assessing user acceptance, ease of use, training requirements, and cultural/organizational factors, organizations can enhance the likelihood of successful adoption and utilization of the system.

3.4 System Specification

3.4.1 Hardware Specification

The hardware specifications required for integrating multi-authority access and encryption for cloud environments depend on the specific implementation and scale of the project. Here are some general hardware requirements that may be necessary:

- **Processor:** Pentium Dual Core 2.00GHz or higher - A powerful processor is essential for efficiently handling the encryption and decryption processes involved in multi-authority access control and encryption.
- **Storage:** Hard Disk - 120 GB or higher - Sufficient storage capacity is crucial for storing encrypted files, access control policies, and system configurations.
- **RAM:** 2GB (minimum) - Sufficient memory (RAM) is essential for efficiently managing encryption keys, authentication tokens, and user sessions within the system.
- **Network Interface:** A reliable network interface is necessary for seamless communication between the multi-authority authentication system, encryption mechanism, and cloud storage environment.
- **Security Hardware:** Hardware-based security features, such as Trusted Platform Modules (TPM) or Hardware Security Modules (HSM), may be incorporated to enhance the security of cryptographic operations and key management processes.

3.4.2 Software Specification

The software requirements for integrating multi-authority access and encryption for cloud environments are critical for ensuring compatibility, security, and functionality. Here are some general software specifications:

- **Operating System:** Windows 7 and above, Linux-based distributions (e.g., Ubuntu, CentOS) - A stable and secure operating system environment suitable for hosting multi-authority access control and encryption software components.
- **Programming Language:** Python 3.x - Python provides a versatile and efficient programming environment for developing and implementing encryption algorithms, access control mechanisms, and cloud storage interfaces.
- **Cloud Storage Interface:** MongoDB Atlas SDK - Integration with MongoDB Atlas, a cloud-based database service, enables seamless interaction with cloud storage resources. This facilitates secure file storage, retrieval, and management while leveraging MongoDB's scalability and reliability features.
- **Database Management System:** MongoDB Atlas - MongoDB Atlas serves as both the cloud storage interface and the database management system, offering a scalable and reliable solution for storing user credentials, access control policies, and audit logs. It supports efficient data management and retrieval operations in a cloud environment.

Language Specification(Python) :

Python 3.6 - Python is selected as the primary programming language for several reasons:

- **Versatility:** Python offers a wide range of libraries and frameworks suitable for various tasks, including encryption, authentication, and cloud storage interaction. Its versatility allows for rapid development and prototyping of complex systems.
- **Readability:** Python's clean and concise syntax promotes readability and maintainability of code, making it easier for developers to collaborate and understand each other's contributions to the project.
- **Rich Ecosystem:** Python boasts a rich ecosystem of third-party libraries and tools, such as PyCryptodome and MongoDB Atlas SDK, which provide robust support for encryption, database interaction, and cloud storage integration. Leveraging these libraries accelerates development and ensures the implementation of secure and efficient solutions.
- **Community Support:** Python has a large and active community of developers, contributors, and enthusiasts who provide extensive documentation, tutorials,

and support resources. Access to this vibrant community fosters innovation, knowledge sharing, and problem-solving throughout the development process.

- **Platform Independence:** Python is platform-independent, allowing the developed software to run seamlessly on various operating systems, including Windows, Linux, and macOS. This ensures compatibility and flexibility in deploying the multi-authority access and encryption system across different environments.

3.4.3 Standards and Policies

- Adheres to industry standards such as GDPR, HIPAA, and SOC 2 for data security and privacy.
- Implements internal policies and procedures to ensure compliance with regulatory requirements.
- Enforces access control policies to manage user permissions and data access.
- Implements encryption standards to protect data both in transit and at rest.
- Maintains a robust incident response plan to address security incidents promptly and effectively.
- Regularly audits and reviews security measures to ensure compliance and effectiveness.
- Provides training and awareness programs for employees to adhere to security policies and best practices.

Python Environment

Python serves as the primary programming language for our project due to its versatility, ease of use, and extensive libraries and frameworks support. It offers a wide range of tools and packages specifically designed for cloud computing, multi-authority access control, and encryption tasks, making it the ideal choice for our project requirements.

Standard Compliance: ISO/IEC 27001

Chapter 4

METHODOLOGY

4.1 General Architecture

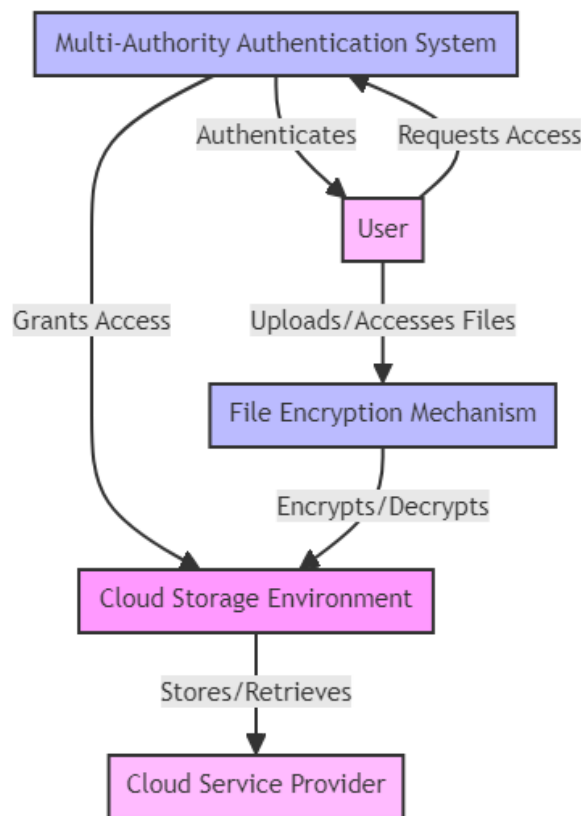


Figure 4.1: **Architecture Diagram**

In this figure 4.1, the architecture diagram illustrates a comprehensive security framework designed for cloud storage environments. This framework consists of several essential components. First, the User represents individuals or entities accessing or storing files in the cloud. Next, the Multi-Authority Authentication System ensures secure user authentication through multiple authorities, enhancing security via diverse authentication methods. Additionally, the File Encryption Mechanism encrypts and decrypts files to safeguard data integrity within the cloud storage system. The Cloud Storage Environment serves as the infrastructure for storing, managing, and retrieving files, typically provided by cloud service providers like

AWS S3 or Google Cloud Storage. Finally, the Cloud Service Provider encompasses the backend infrastructure and services supporting the cloud storage environment, including physical servers, networking, and data management software. This architecture diagram aims to illustrate the interconnectedness of these components in fortifying cloud storage systems against unauthorized access and data breaches.

4.2 Design Phase

4.2.1 Data Flow Diagram

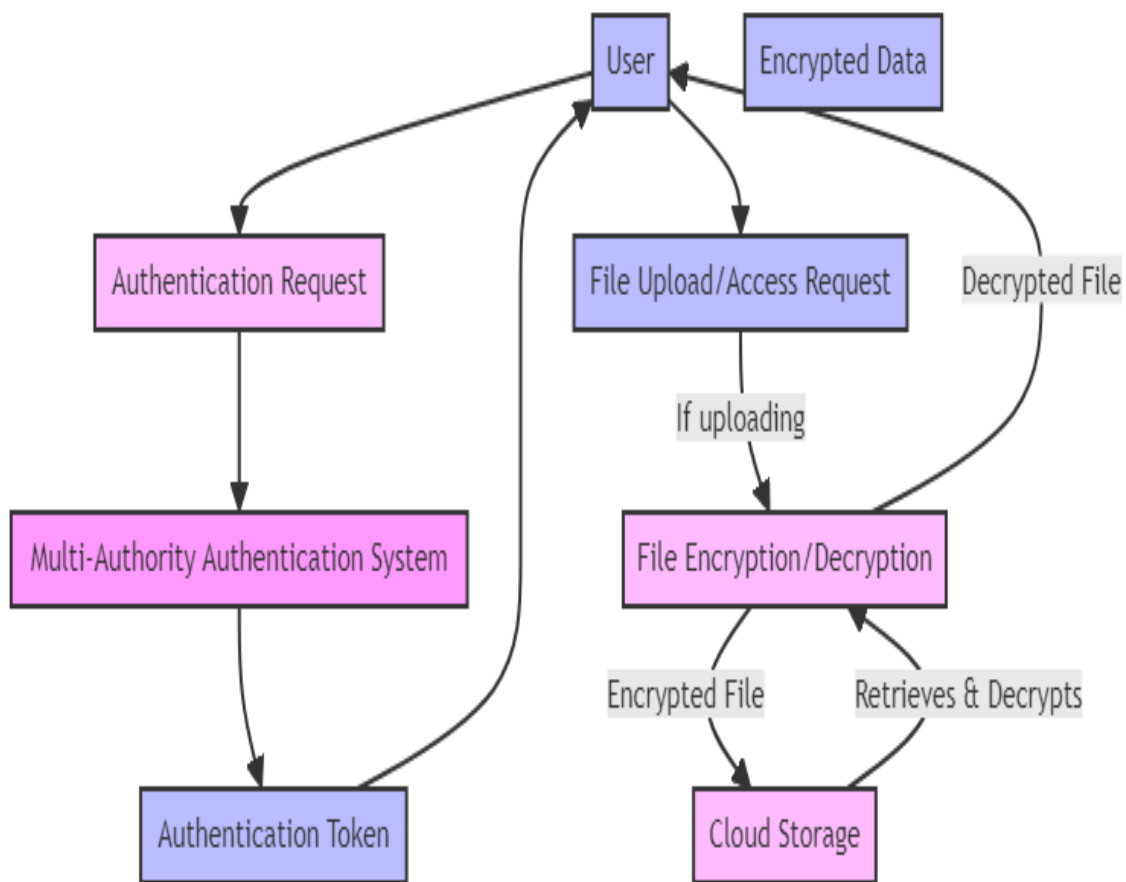


Figure 4.2: Data flow diagram

In this Figure 4.2, the data flow diagram illustrates the seamless interaction between various components within a cloud storage security framework. It begins with the User initiating actions such as authentication requests or file uploads/downloads to access the cloud storage service. Authentication Requests are then processed by the Multi-Authority Authentication System, utilizing multiple verification methods to confirm user credentials securely. Upon successful authentication, users receive

Authentication Tokens, representing their verified access rights. Subsequently, users proceed with File Upload/Access Requests, interacting with the File Encryption/Decryption mechanism, which encrypts files before storage and decrypts them upon retrieval to maintain data integrity. Encrypted data is stored within the Cloud Storage environment, ensuring data confidentiality and preventing unauthorized access. This diagram highlights the efficient flow of information and actions, emphasizing user authentication and data security within the cloud storage system.

4.2.2 Use Case Diagram

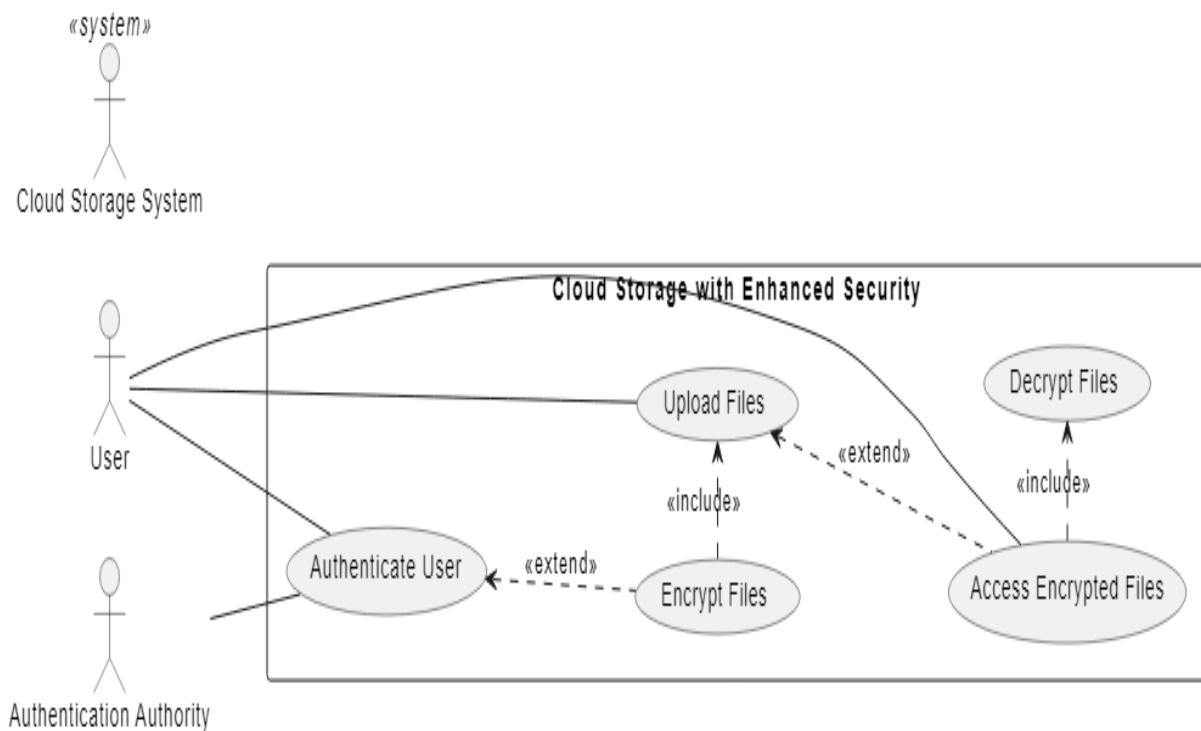


Figure 4.3: Use case diagram

In this Figure 4.3, the Use Case Diagram provides a visual representation of the interactions between users, administrators, and the system in the integration of a Multi-Authority Authentication System with a File Encryption Mechanism for enhanced security in Cloud Storage Environments. The diagram delineates various actors, including users and administrators, along with the system itself. Through use cases, specific interactions such as user authentication, file encryption, access control, and system configuration are depicted. This diagram offers a clear overview of how different stakeholders interact with the system and helps in identifying key functionalities and requirements. It serves as a valuable tool for understanding the system's

behavior and functionality in a structured manner, facilitating effective communication among project stakeholders and aiding in the development and implementation of secure cloud storage solutions.

4.2.3 Class Diagram

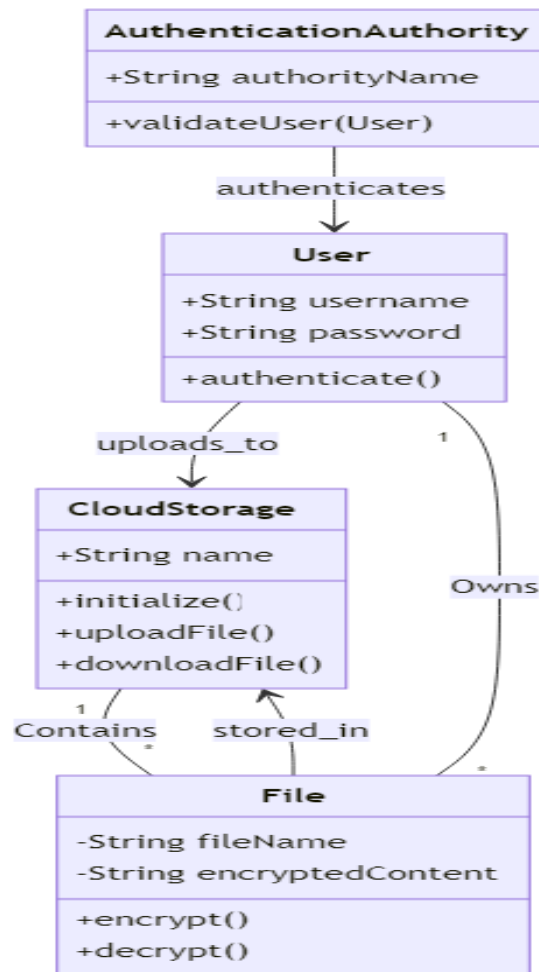


Figure 4.4: Class diagram

In this Figure 4.4, In addition to depicting classes like User, AuthenticationSystem, File, EncryptionMechanism, CloudStorage, and Authority, a class diagram also outlines the attributes and methods associated with each class, providing a comprehensive view of their functionalities. For instance, the User class may have attributes such as username and password, along with methods for authentication. The AuthenticationSystem class might include methods for verifying user credentials with multiple authorities. Similarly, the EncryptionMechanism class could have attributes

representing encryption algorithms and methods for encrypting and decrypting files. Furthermore, the relationships depicted in the class diagram illustrate how these classes collaborate to ensure secure authentication, encryption, and storage of files in the cloud. This detailed representation aids in understanding the underlying structure of the system and facilitates effective design and implementation of the integration between the authentication system and encryption mechanism for enhanced security in cloud storage environments.

4.2.4 Sequence Diagram

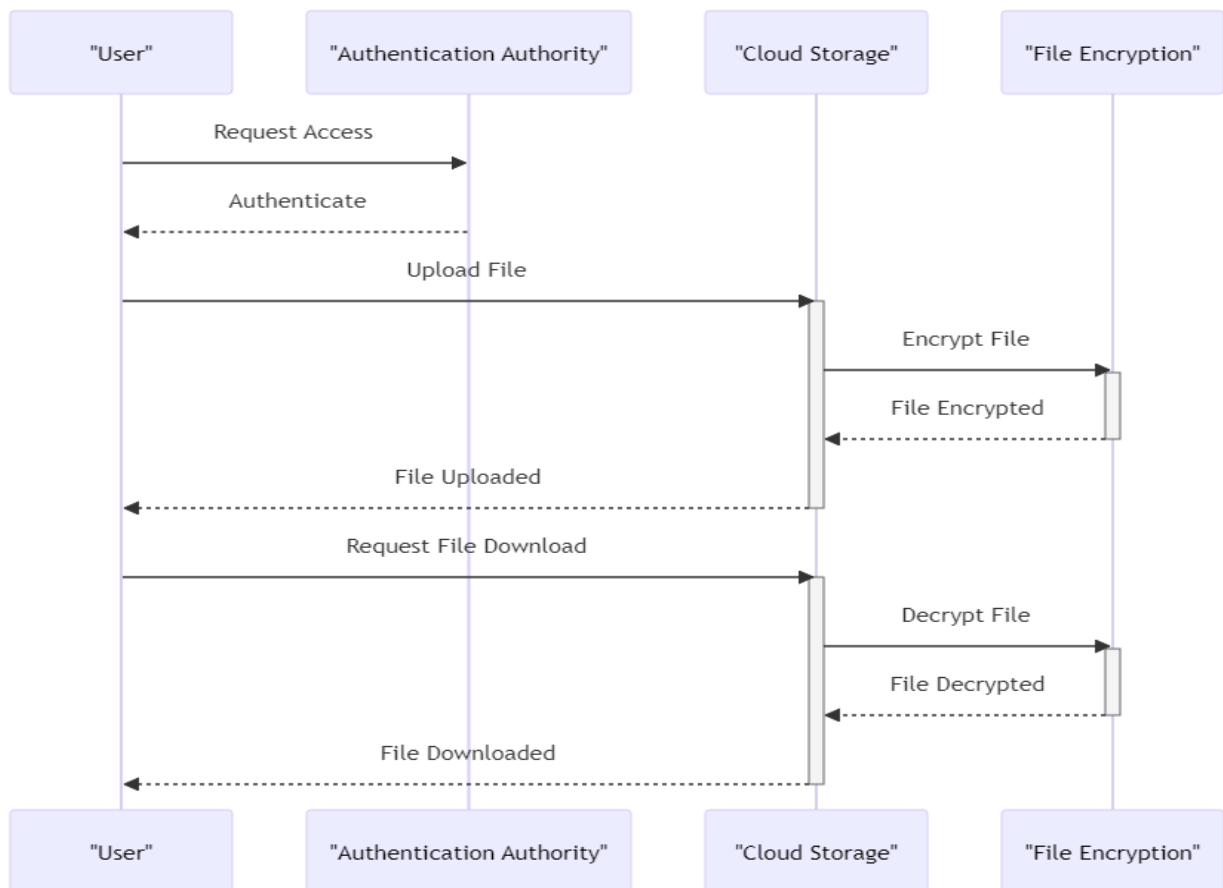


Figure 4.5: Sequence diagram

In this Figure 4.5, the visual representation offers a detailed insight into the orchestrated series of actions required to uphold data security within cloud storage environments. From the outset of transmitting authentication details to their rigorous validation by multiple authorities, the diagram exemplifies the meticulousness

essential for robust security measures. Moreover, it vividly illustrates the encryption process, showcasing how files undergo transformation into an encrypted state before being securely stored in the cloud, safeguarding sensitive information against unauthorized access. Additionally, the diagram elucidates the nuanced management of access rights, illustrating the dynamic interplay between users, administrators, and security components, thereby ensuring granular control over data access privileges. This comprehensive portrayal not only facilitates a profound understanding of the interdependencies among security measures but also fosters effective communication and collaboration among stakeholders, paving the way for the development of resilient cloud storage security solutions.

4.2.5 Activity Diagram

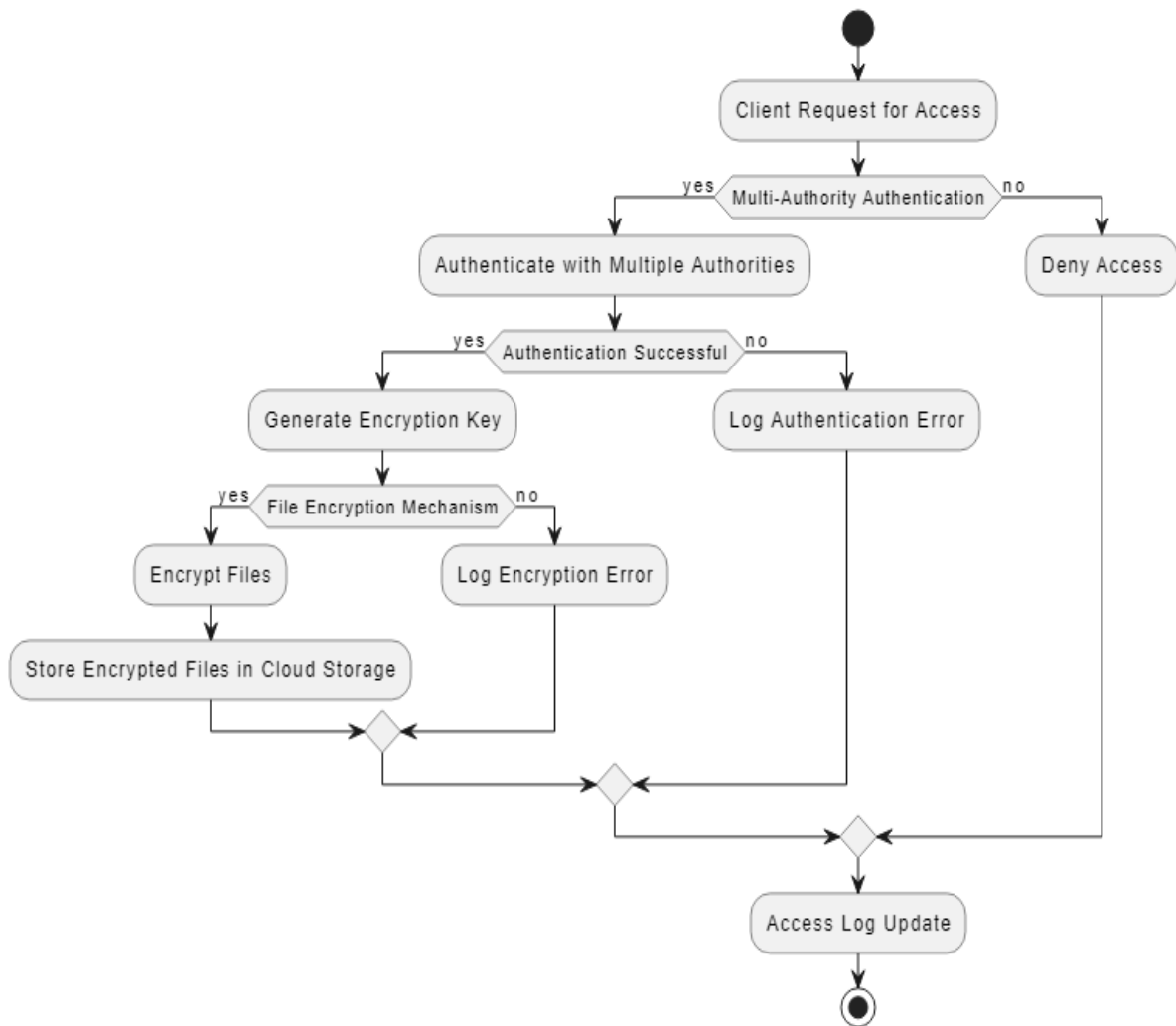


Figure 4.6: Activity diagram

In this Figure 4.6, the Activity Diagram vividly portrays the dynamic interactions between users, authorities, and system components, offering invaluable insights into the coordinated efforts required to safeguard data in cloud storage environments. Each sequential step is meticulously detailed, underscoring the pivotal role of user authentication as the initial gateway to access the system securely. Notably, the parallel verification processes underscore the multi-authority authentication system's effectiveness in fortifying security through redundancy and diversity in authentication methods. As files progress through the encryption phase, the diagram underscores the paramount importance of preserving data integrity during transit and storage, thereby ensuring that sensitive information remains inaccessible to unauthorized entities. This meticulous visualization not only facilitates a deeper comprehension of the security mechanisms at play but also serves as a crucial guiding framework for stakeholders, empowering them to identify potential vulnerabilities and implement necessary enhancements effectively. Ultimately, the Activity Diagram serves as a cornerstone in the ongoing evolution of cloud storage security solutions, providing organizations with the tools and insights needed to proactively mitigate risks and safeguard their valuable data assets.

4.3 Algorithm & Pseudo Code

4.3.1 Algorithm Used: Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES) is a symmetric encryption algorithm widely adopted for securing data in various applications, including cloud computing. In our project, we utilize AES to implement encryption and decryption functionalities for protecting data stored in cloud environments.

Key Components:

Key Expansion: AES operates on fixed block sizes of 128 bits and supports key sizes of 128, 192, or 256 bits. The key expansion process generates a set of round keys from the initial encryption key, which are used in the encryption and decryption rounds.

Substitution: AES employs a substitution-permutation network (SPN) structure, where each byte of data is substituted with another byte according to a predefined substitution table (S-box). This step enhances the confusion property of the encryption process.

Permutation: Permutation involves rearranging the bytes of data in a specific pattern, which further obscures the relationship between the plaintext and ciphertext. This step contributes to the diffusion property of the encryption process.

Rounds: AES operates multiple rounds of substitution, permutation, and other operations depending on the key size. Each round modifies the state of the data to create increasingly scrambled ciphertext.

Advantages :

Security: AES offers a high level of security against various cryptographic attacks due to its strong encryption mechanism and large key space. It has been extensively analyzed and validated by cryptographic experts worldwide.

Performance: AES is computationally efficient and can be implemented efficiently on a wide range of hardware and software platforms. It provides a good balance between security and performance, making it suitable for real-world applications.

Standardization: AES is a standard encryption algorithm recommended by government agencies and industry standards organizations, ensuring interoperability and compatibility across different systems and platforms.

4.3.2 Pseudo Code

AES Encryption Algorithm

Step 1: Key Expansion (Key Schedule Generation)

- The AES algorithm starts with expanding the original key into a key schedule.
- This key expansion process generates a series of round keys from the original encryption key.
- Each round key is used in the main encryption process for each round.

Step 2: Initial Round

- In the initial round, the plaintext is combined with the first round key using a bitwise XOR operation.
- This step is known as "AddRoundKey."

Step 3: Main Rounds

- The main encryption process consists of multiple rounds (Nr rounds, where Nr depends on the key size).
- Each round performs four transformations: SubBytes, ShiftRows, MixColumns, and AddRoundKey.
- **SubBytes:** Each byte in the state matrix is replaced with a corresponding value from a substitution table (S-box).
- **ShiftRows:** The rows of the state matrix are shifted cyclically to the left. The first row remains unchanged, the second row is shifted by one position to the left, the third row by two positions, and the fourth row by three positions.
- **MixColumns:** Each column of the state matrix is transformed using a matrix multiplication operation. This step provides diffusion in the data.
- **AddRoundKey:** Each byte of the state matrix is combined with the corresponding byte of the round key using a bitwise XOR operation.

Step 4: Final Round

- The final round is similar to the main rounds but excludes the MixColumns step.
- It consists of SubBytes, ShiftRows, and AddRoundKey operations using the final round key.

Output

- After completing all the rounds, the resulting state matrix represents the encrypted ciphertext. The plaintext is encrypted using the AES algorithm. It provides confidentiality and security by transforming the input data through a series of substitution, permutation, and mixing operations. The key expansion ensures that each round uses a different round key, adding complexity to the encryption process and enhancing security.

4.4 Steps to execute/run/implement the project

4.4.1 Step 1: Requirement Analysis:

- Understand security needs and operational requirements for cloud storage.
- Define authentication, access control, and encryption requirements.

4.4.2 System Design:

- Architect the integration of multi-authority authentication and file encryption mechanisms.
- Design interactions between components and cloud storage environment.

4.4.3 Select Technologies:

- Choose appropriate frameworks and libraries for authentication, encryption, and cloud storage services.

4.4.4 Implement Multi-Authority Authentication:

- Develop user registration, authentication processes, and access control policies.
- Implement handling of authentication requests from multiple authorities.

4.4.5 File Encryption Implementation:

- Develop encryption mechanism using algorithms like AES.
- Generate and manage unique encryption keys for each file.

4.4.6 Integration with Cloud Storage:

- Develop APIs or connectors to interact securely with cloud storage services.
- Integrate authentication and encryption mechanisms with chosen cloud storage provider (e.g., AWS S3, Google Cloud Storage).

4.4.7 Testing:

- Conduct comprehensive testing of authentication flows, encryption processes, and interactions with cloud storage.
- Ensure functionality, security, and performance meet requirements.

4.4.8 Security Review:

- Perform thorough security review to identify and address vulnerabilities.
- Ensure compliance with security best practices and standards.

4.4.9 Deployment:

- Deploy integrated system in cloud environment.
- Configure settings and permissions for secure operation.

4.4.10 Monitoring and Maintenance:

- Implement monitoring mechanisms for system performance and security events.
- Establish procedures for regular maintenance, updates, and security audits.

These steps provide a structured approach to implementing the integration of multiauthority authentication with file encryption for enhanced security in cloud storage environments.

Chapter 5

IMPLEMENTATION AND TESTING

5.1 Input and Output

5.1.1 Input Design

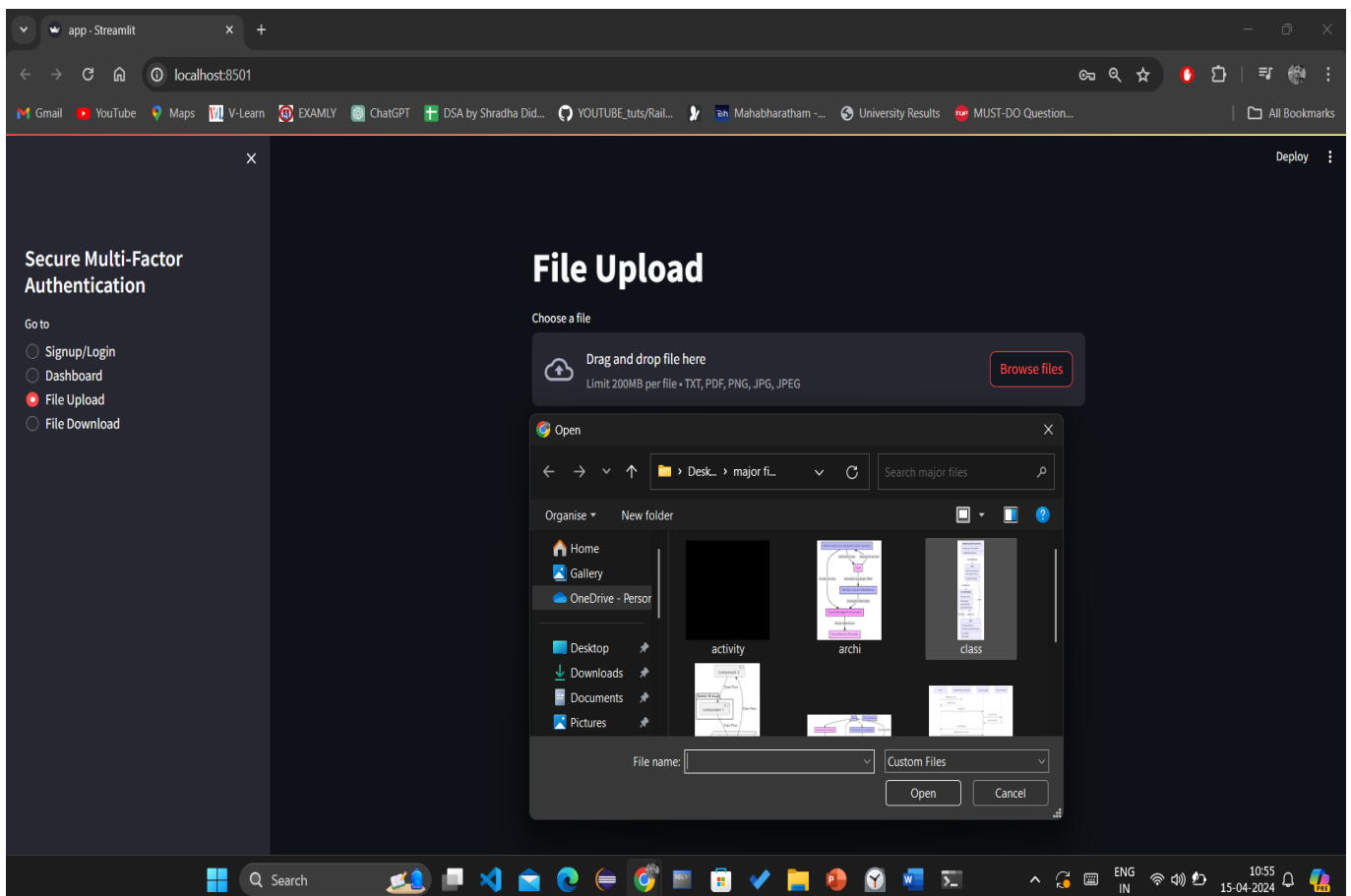


Figure 5.1: File Selection and upload Window

5.1.2 Implement Multi-Authority Authentication:

In the figure 5.1, Input design is a crucial phase in the development of any software system, ensuring that data is captured accurately, efficiently, and securely. Here's an outline of content that could be included in input design:

1. Data Entry Forms:

- Design intuitive and user-friendly forms for data entry.

- Organize fields logically and group related information together.
- Use appropriate input controls such as text fields, dropdown menus, radio buttons, and checkboxes.
- Provide clear labels and instructions to guide users through the data entry process.

2. Data Validation:

- Implement validation rules to ensure that entered data meets specified criteria (e.g., data type, format, range).
- Display error messages to alert users of any validation failures and provide guidance on how to correct them.
- Perform both client-side and server-side validation to prevent invalid data from being submitted.

3. Data Verification:

- Include mechanisms for verifying the accuracy of entered data through double entry or cross-referencing with existing records.
- Implement validation checks to identify duplicate or inconsistent data entries.
- Provide tools for data reconciliation and resolution of discrepancies.

4. Data Security:

- Incorporate measures to protect sensitive data during input, transmission, and storage.
- Implement authentication and authorization mechanisms to control access to data entry forms and prevent unauthorized modifications.
- Encrypt sensitive information to safeguard it from unauthorized access or interception.

5. Input Controls:

- Use input controls and data formatting options to minimize errors and improve data quality.
- Implement features such as autocomplete, dropdown lists, and date pickers to streamline data entry and reduce manual input errors.

- Provide input masks to enforce specific data formats (e.g., phone numbers, social security numbers) and guide users during data entry.

6. Error Handling:

- Design error handling mechanisms to gracefully handle input errors and prevent system crashes.
- Provide informative error messages that clearly explain the nature of the error and suggest possible solutions.
- Include options for users to review and correct their input before resubmitting.

7. Accessibility:

- Ensure that data entry forms are accessible to users with disabilities by following accessibility guidelines and standards.
- Use semantic HTML markup, ARIA attributes, and keyboard navigation support to enhance accessibility.
- Provide alternative input methods (e.g., voice input) for users who may have difficulty with traditional input devices.

8. Usability Testing:

- Conduct usability testing to evaluate the effectiveness and efficiency of data entry forms.
- Gather feedback from users to identify pain points, usability issues, and areas for improvement.
- Iterate on the design based on user feedback to optimize the user experience.

5.1.3 Output Design

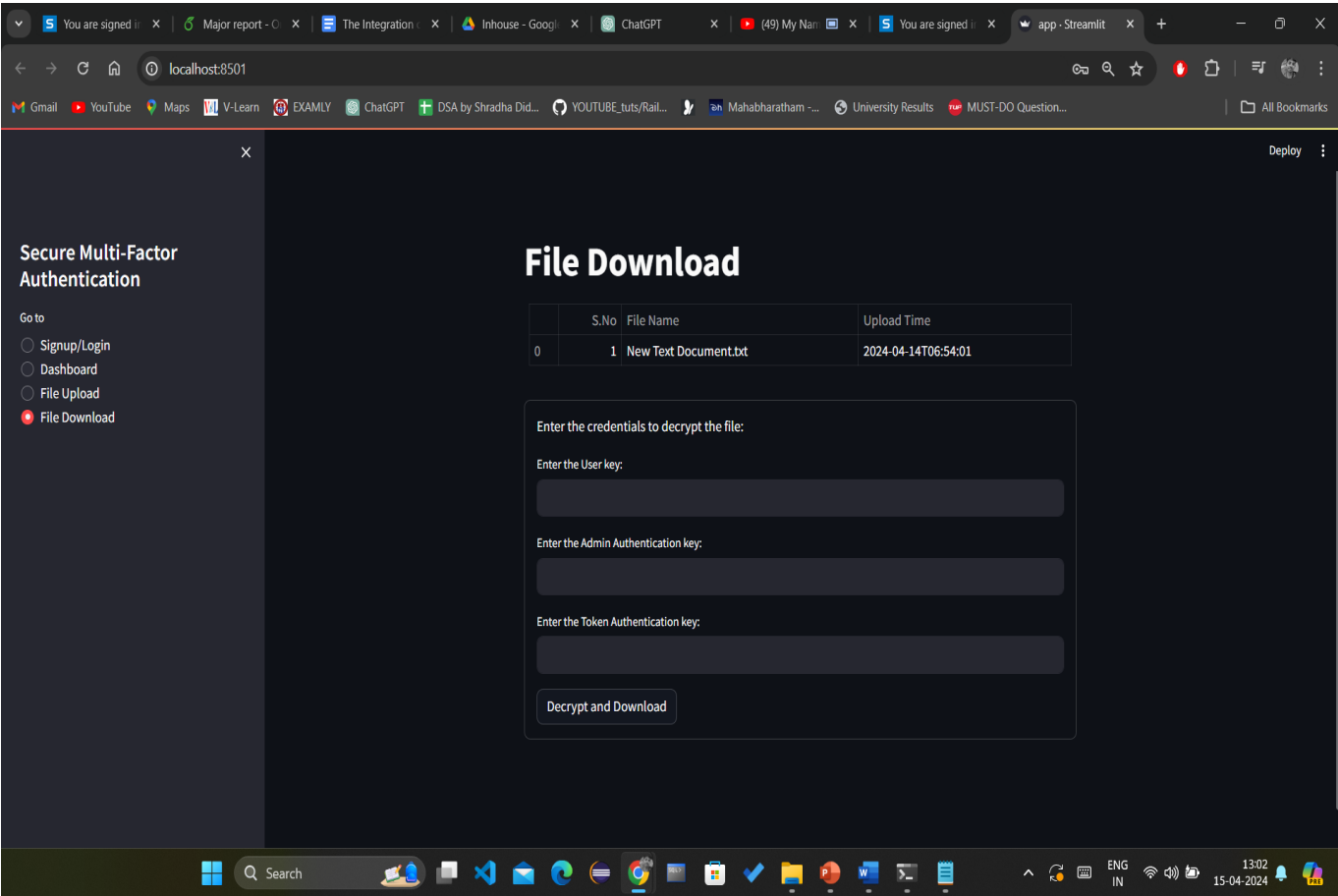


Figure 5.2: File Encrypted and uploaded to Cloud

If the above Figure 5.2, portrays a confirmation message or visual indicator denoting the successful upload of files. This confirmation may include textual affirmation of the upload’s success, accompanied by pertinent icons or symbols such as a checkmark or progress bar, indicating the completion of the upload process. Moreover, the image could also incorporate elements indicating the destination of the uploaded files, such as the name or logo of the cloud storage platform or file-sharing service. Additionally, it might include details regarding the uploaded files, such as the number of files uploaded, their sizes, or any relevant timestamps. Overall, the image serves to reassure users that their file uploads have been completed successfully and provides pertinent information regarding the uploaded files for reference or verification purposes.

5.2 Testing

Discovering and fixing such problems is what testing is all about. The purpose of testing is to find and correct any problems with the final product. It's a method for evaluating the quality of the operation of anything from a whole product to a single component. The goal of stress testing software is to verify that it retains its original functionality under extreme circumstances. There are several different tests from which to pick. Many tests are available since there is such a vast range of assessment options. All individuals who play an integral role in the software development process are responsible for performing the testing. Testing the software is the responsibility of a wide variety of specialists, including the End Users, Project Manager, Software Tester, and Software Developer.

Testing the software is the initial step in the process. begins with the phase of requirement collecting, also known as the Planning phase, and ends with the stage known as the Deployment phase. In the waterfall model, the phase of testing is where testing is explicitly arranged and carried out. Testing in the incremental model is carried out at the conclusion of each increment or iteration, and the entire application is examined in the final test. Testing the programme is an ongoing activity that will never end. Without first putting the software through its paces, it is impossible for anyone to guarantee that it is completely devoid of errors. Because the domain to which the input belongs is so expansive, we are unable to check every single input.

5.3 Types of Testing

5.3.1 Unit Testing

Unit testing for the integration of a Multi-Authority Authentication System with a File Encryption Mechanism in Cloud Storage environments involves testing each individual component of the system to ensure they function correctly when integrated. Three test cases for this scenario could be:

Test case 1: Verify that the Multi-Authority Authentication System correctly authenticates multiple users accessing the cloud storage.

Test case 2: Test the File Encryption Mechanism to ensure that files are encrypted and decrypted accurately without data loss or corruption.

Test case 3: Validate the integration of the Authentication System with the Encryp-

tion Mechanism to confirm that only authorized users can access and decrypt encrypted files in the cloud storage environment.

By conducting thorough unit testing with these test cases, developers can identify and fix any issues early in the development process, ensuring a more secure and reliable system for cloud storage environments.

Input

```
1 import json
2 import os
3 import numpy as np
4 import streamlit as st
5 from streamlit import session_state
6 import pdfplumber
7 import docx
8 import pyaes
9 import random
10 import pandas as pd
11 import base64
12 import hashlib
13 from pymongo import MongoClient
14 import smtplib
15 import string
16 import re
17 import datetime
18
19 from dotenv import load_dotenv
20 load_dotenv()
21
22 ss = st.session_state
23 if "index" not in ss:
24     ss["index"] = 0
25
26
27 @st.cache_resource
28 def init_connection():
29     return MongoClient(
30         os.getenv("MONGO_URI")
31     )
32
33 client = init_connection()
34 db = client["file_data"]
35 users_coll = db["users"]
36 print("Connected to MongoDB!")
37 print(users_coll)
38
39 def user_present(email):
40     try:
41         user = users_coll.find_one({"email": email})
42         if user:
43             return True
44         return False
45     except Exception as e:
46         st.error(f"Error checking user: {e}")
47         return False
48
49
50 def send_code(email, code):
51     SENDER_EMAIL = os.getenv("SENDER_EMAIL")
52     APP_PASSWORD = os.getenv("APP_PASSWORD")
53     RECEIVER = email
54     server = smtplib.SMTP_SSL("smtp.googlemail.com", 465)
55     server.login(SENDER_EMAIL, APP_PASSWORD)
56     message = f"Subject: Verification Code\n\nYour code is: {code}"
57     server.sendmail(SENDER_EMAIL, RECEIVER, message)
58     server.quit()
59     st.success("Email sent!")
60     return True
```

```

61
62
63 def generate_code(length=6):
64     return "".join(random.choices(string.ascii_uppercase + string.digits, k=length))
65
66
67 def signup(json_file_path="students.json"):
68     st.title("Student Signup")
69     with st.form("signup_form"):
70         st.write("Fill in the details to create an account:")
71         name = st.text_input("Name:")
72         email = st.text_input("Email:")
73         age = st.number_input("Age:", min_value=0, max_value=120)
74         sex = st.radio("Sex:", ("Male", "Female", "Other"))
75         password = st.text_input("Password:", type="password")
76         confirm_password = st.text_input("Confirm Password:", type="password")
77         if (
78             ss.get("verification_code") is None
79             or ss.get("verification_time") is None
80             or datetime.datetime.now() - ss.get("verification_time")
81             > datetime.timedelta(minutes=5)
82         ):
83             code = generate_code()
84             ss["verification_code"] = code
85             ss["verification_time"] = datetime.datetime.now()
86         if st.form_submit_button("Signup"):
87             if not name:
88                 st.error("Name cannot be empty.")
89             elif not email:
90                 st.error("Email cannot be empty.")
91             elif not re.match(r"^[w\.-]+@[w\.-]+$", email):
92                 st.error("Invalid email format. Please enter a valid email address.")
93             elif user_present(email):
94                 st.error("User with this email already exists.")
95             elif not age:
96                 st.error("Age cannot be empty.")
97             elif not password or len(password) < 6:
98                 st.error("Password must be at least 6 characters long.")
99             elif password != confirm_password:
100                 st.error("Passwords do not match. Please try again.")
101             else:
102                 code = ss["verification_code"]
103                 send_code(email, code)
104                 entered_code = st.text_input("Enter the verification code:")
105                 if entered_code == code:
106                     user = create_account(name, email, age, sex, password)
107                     ss["logged_in"] = True
108                     ss["user_info"] = user
109                     st.success("Signup successful. You are now logged in!")
110                 elif len(entered_code) == 6 and entered_code != code:
111                     st.error("Incorrect code. Please try again.")
112
113
114 def login():
115     st.title("Login")
116     username = st.text_input("Email:")
117     password = st.text_input("Password:", type="password")
118     username = username.lower()
119     password = hashlib.sha256(password.encode()).hexdigest()
120
121     if st.button("Login"):
122         user = check_login(username, password)
123         if user:
124             ss["logged_in"] = True
125             ss["user_info"] = user
126         else:
127             st.error("Invalid credentials. Please try again.")
128
129 def render_dashboard(user_info):
130     try:
131         st.title(f"Welcome, {user_info['name']}!")
132         st.subheader("User Information:")
133         st.write(f"Name: {user_info['name']}")
134         st.write(f"Sex: {user_info['sex']}")
135         st.write(f"Age: {user_info['age']}")

```

```

135
136 except Exception as e:
137     st.error(f"Error rendering dashboard: {e}")
138
139
140 def generate_key(user_key, admin_auth, token_auth):
141     key = hashlib.sha256(
142         user_key.encode("utf-8")
143         + admin_auth.encode("utf-8")
144         + token_auth.encode("utf-8")
145     ).digest()[:16]
146     return key
147
148
149 def extract_text(file) -> str:
150     if isinstance(file, str):
151         ext = os.path.splitext(file)[1].lower()
152     else:
153         ext = os.path.splitext(file.name)[1].lower()
154
155     if ext == ".pdf":
156         if isinstance(file, str):
157             with pdfplumber.open(file) as pdf:
158                 text = "\n".join(
159                     page.extract_text() for page in pdf.pages if page.extract_text()
160                 )
161         else:
162             with pdfplumber.open(file) as pdf:
163                 text = "\n".join(
164                     page.extract_text() for page in pdf.pages if page.extract_text()
165                 )
166     elif ext == ".docx":
167         if isinstance(file, str):
168             doc = docx.Document(file)
169         else:
170             doc = docx.Document(file)
171         text = "\n".join([para.text for para in doc.paragraphs])
172     else:
173         if isinstance(file, str):
174             with open(file, "r", encoding="utf-8", errors="ignore") as f:
175                 text = f.read()
176         else:
177             with file as f:
178                 text = f.read()
179     return text

```

Test result

After successfully uploading files, users are typically presented with a confirmation message or visual indicator, reassuring them of the completion of the transfer process. This confirmation instills confidence in users, affirming that their files have been securely transmitted to the intended destination. Additionally, the presence of visual cues such as checkmarks or progress bars enhances user experience by providing clear and intuitive feedback on the upload progress. This confirmation message not only eliminates any uncertainty regarding the status of the upload but also contributes to a seamless and efficient user experience, fostering trust in the reliability and functionality of the file transfer process.

5.3.2 Integration Testing

Integration testing for the integration of a multi-authority authentication system with a file encryption mechanism in cloud storage environments ensures that all components interact correctly and secure data effectively. This testing phase verifies the seamless communication between the authentication system and the encryption mechanism to provide enhanced security for cloud storage.

Testcase1: Validate that user authentication from the multi-authority system successfully grants access to encrypt and decrypt files in the storage environment.

Testcase2: Confirm that encrypted files can only be decrypted with the correct authentication credentials from the multi-authority system.

Testcase3: Simulate a scenario where an unauthorized user attempts to access and decrypt files without proper authentication, ensuring that the encryption mechanism effectively blocks access.

By executing these test cases, the integration of the authentication system with the encryption mechanism can be thoroughly evaluated to ensure the security and functionality of the cloud storage environment.

Input

```
1 def get_keys():
2     with st.form("credentials"):
3         st.write("Enter credentials to encrypt the file:")
4         user_key = st.text_input("User key:")
5         admin_auth = st.text_input("Admin Authentication key:")
6         token_auth = st.text_input("Token Authentication key:")
7         if st.form_submit_button("Encrypt and Upload"):
8             return user_key, admin_auth, token_auth
9     return None, None, None
10
11
12 def main(
13     json_file_path="data.json",
14 ):
15     users_coll = db["users"]
16     st.sidebar.title("Secure Multi-Factor Authentication")
17     page = st.sidebar.radio(
18         "Go to",
19         (
20             "Signup/Login",
21             "Dashboard",
22             "File Upload",
23             "File Download",
24         ),
25         key="Secure Multi-Factor Authentication",
26     )
27
28     if page == "Signup/Login":
29         st.title("Signup/Login")
30         login_or_signup = st.radio(
31             "Select an option", ("Login", "Signup"), key="login_signup"
32         )
33         if login_or_signup == "Login":
34             login()
35         else:
```

```

36         signup()
37
38     elif page == "Dashboard":
39         if ss.get("logged_in"):
40             render_dashboard(ss["user_info"])
41         else:
42             st.warning("Please login/signup to view the dashboard.")
43
44     elif page == "File Upload":
45         if ss.get("logged_in"):
46             st.title("File Upload")
47             uploaded_file = st.file_uploader(
48                 "Choose a file", type=["txt", "pdf", "png", "jpg", "jpeg"]
49             )
50             if uploaded_file:
51                 file_details = {
52                     "filename": uploaded_file.name,
53                     "filetype": uploaded_file.type,
54                     "filesize": uploaded_file.size,
55                 }
56                 st.write("Name: %s" % uploaded_file.name)
57                 st.write("Type: %s" % uploaded_file.type)
58                 st.write("Size: %s" % uploaded_file.size)
59                 st.write("Enter credentials to encrypt the file:")
60                 with st.form("credentials"):
61                     st.write("Enter credentials to encrypt the file:")
62                     user_key = st.text_input("User key:")
63                     admin_auth = st.text_input("Admin Authentication key:")
64                     token_auth = st.text_input("Token Authentication key:")
65                     if st.form_submit_button("Encrypt and Upload"):
66                         if users_coll.find_one(
67                             {"email": ss["user_info"]["email"]}
68                         ):
69                             user_info = users_coll.find_one(
70                                 {"email": ss["user_info"]["email"]}
71                             )
72                             if user_info["files"] is None:
73                                 user_info["files"] = []
74                             key = generate_key(user_key, admin_auth, token_auth)
75                             aes = pyaes.AESModeOfOperationCTR(key)
76                             file = base64.b64encode(uploaded_file.read()).decode(
77                                 "utf-8"
78                             )
79                             cipher_text = aes.encrypt(file)
80                             cipher_text = base64.b64encode(cipher_text).decode("utf-8")
81                             current_time = str(np.datetime64("now"))
82                             file_name = uploaded_file.name
83                             for f in user_info["files"]:
84                                 if f["file"] == file_name:
85                                     file_name = (
86                                         file_name.split(".")[0]
87                                         + "_1."
88                                         + file_name.split(".")[1]
89                                     )
90                             user_info["files"].append(
91                                 {
92                                     "file": uploaded_file.name,
93                                     "data": cipher_text,
94                                     "time": current_time,
95                                     "sanitized": False,
96                                 }
97                             )
98                             users_coll.update_one(
99                                 {"email": ss["user_info"]["email"]},
100                                 {"$set": {"files": user_info["files"]}},
101                             )
102                             st.success("File uploaded!")
103             else:
104                 st.warning("Please login/signup.")
105
106     elif page == "File Download":
107         if ss.get("logged_in"):
108             st.title("File Download")
109             user_info = users_coll.find_one(

```



```

110         {"email": ss["user_info"]["email"]}]
111     )
112     if user_info["files"]:
113         i = 1
114         files = []
115         for f in user_info["files"]:
116             file_data = {}
117             file_data["S.No"] = i
118             file_data["File Name"] = f["file"]
119             file_data["Upload Time"] = f["time"]
120             files.append(file_data)
121             i += 1
122         st.table(files)
123         try:
124             with st.form("credentials2"):
125                 st.write("Enter credentials to decrypt the file:")
126                 user_key = st.text_input("User key:")
127                 admin_auth = st.text_input("Admin Authentication key:")
128                 token_auth = st.text_input("Token Authentication key:")
129                 if st.form_submit_button("Decrypt and Download"):
130                     key = generate_key(user_key, admin_auth, token_auth)
131                     aes = pyaes.AESModeOfOperationCTR(key)
132                     file_name = st.text_input("Enter the file name to download:")
133                     for f in user_info["files"]:
134                         if f["file"] == file_name:
135                             data = base64.b64decode(f["data"])
136                             decrypted_text = aes.decrypt(data).decode("utf-8")
137                             data = base64.b64decode(decrypted_text)
138                             with open(file_name, "wb") as file:
139                                 file.write(data)
140                             st.success("File downloaded!")
141             except Exception as e:
142                 st.error(f"Wrong credentials")
143         else:
144             st.warning("Please login/signup.")

```

Test result

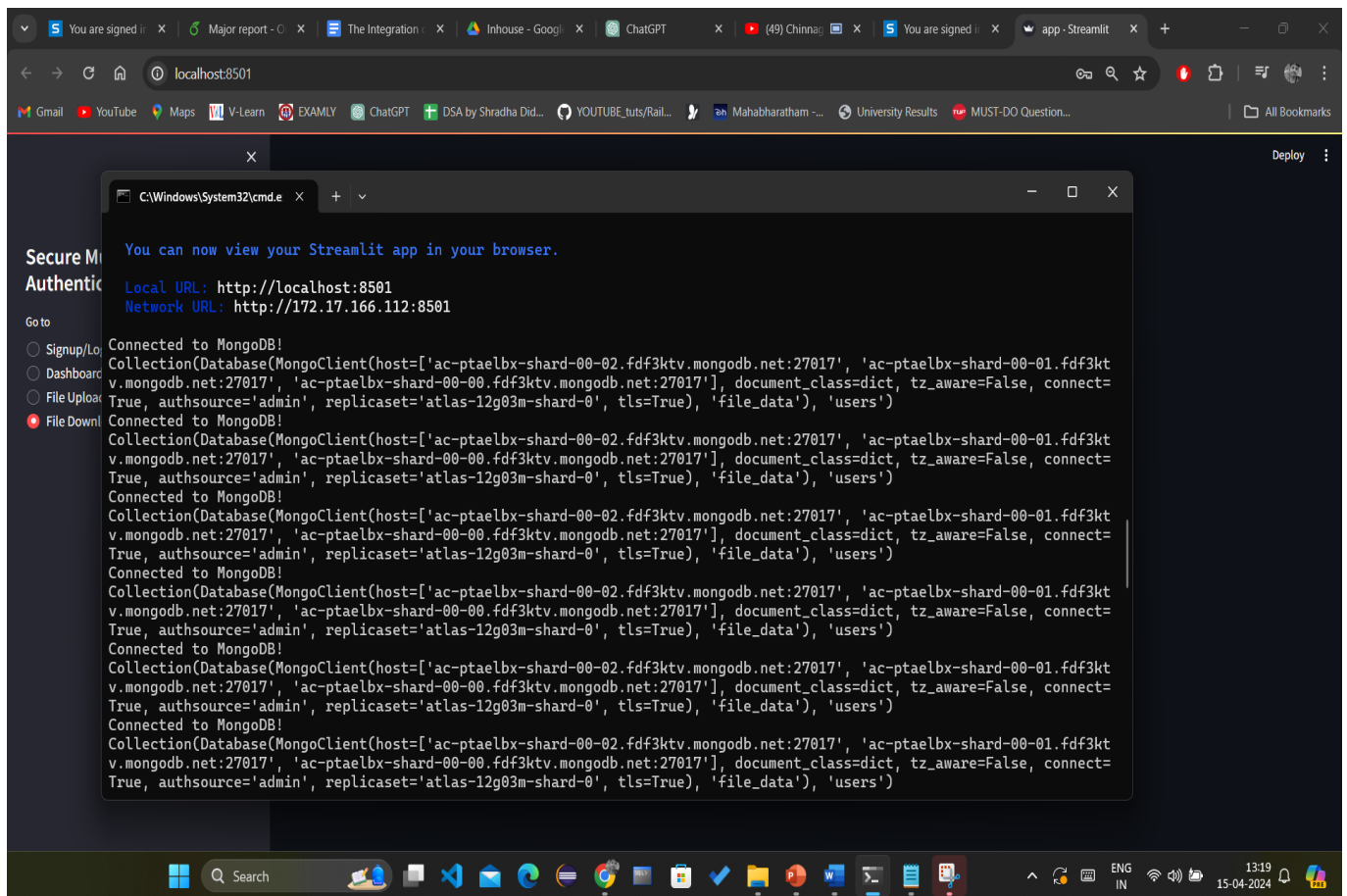


Figure 5.3: **Successful connection of Database**

In the above figure 5.3, the Integration testing for reversible data hiding in images can be presented in various formats, such as tables, graphs, and visualizations. The results can also be compared with those of other algorithms or state-of-the-art techniques to determine the effectiveness and efficiency of the proposed algorithm.

In summary, the test results for reversible data hiding in images are critical to evaluating the performance of the algorithm and determining its suitability for various applications. The results should be presented in a clear and comprehensive manner to facilitate their interpretation and comparison.

5.3.3 System Testing

System testing will focus on evaluating the following aspects of the integrated multi-authority access and encryption system for cloud environments:

- **Functionality:** Testing the system's features and functionalities such as user authentication, access control enforcement, encryption, decryption, and data storage operations.
- **Performance:** Evaluating the system's responsiveness, scalability, and resource utilization under different workloads and conditions.
- **Security:** Identifying and addressing vulnerabilities, threats, and risks associated with the system's security mechanisms and implementations.
- **Usability:** Assessing the system's user interface, user experience, and overall ease of use to ensure user satisfaction.

Result :

System testing plays a critical role in ensuring the reliability, performance, and security of the integrated multi-authority access and encryption system for cloud environments. By conducting comprehensive testing across different dimensions, potential issues and risks are identified and addressed, leading to a robust and reliable system that meets user requirements and expectations.

5.3.4 Test Result

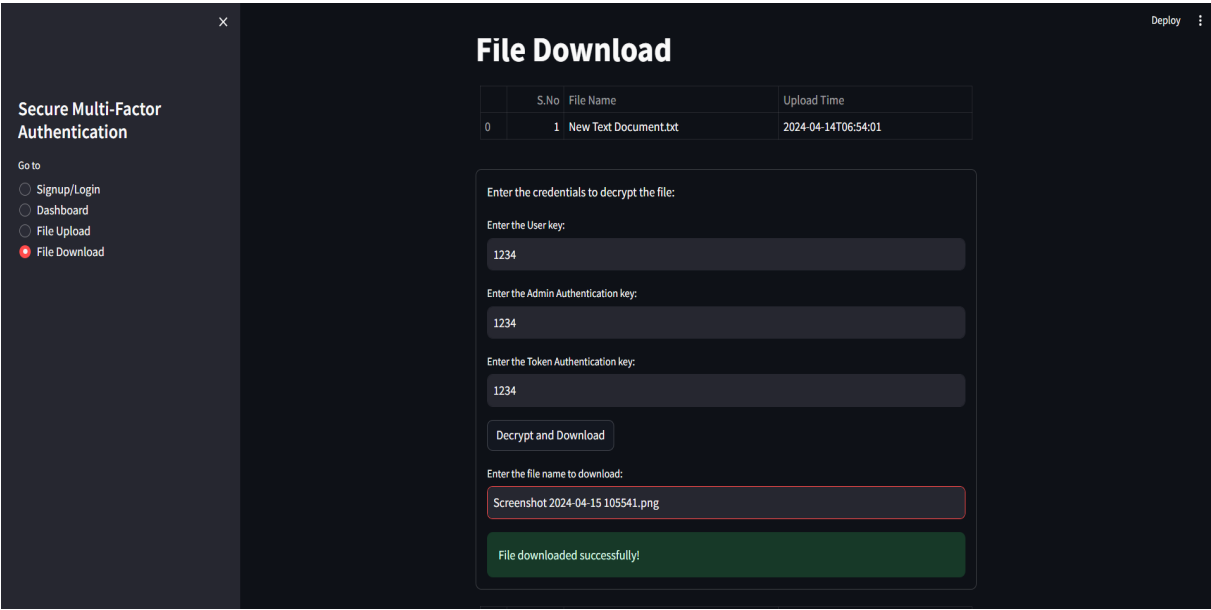


Figure 5.4: GUI Home Window

In the figure 5.4, After successfully uploading files, users are typically presented with a confirmation message or visual indicator, reassuring them of the completion of the transfer process. This confirmation instills confidence in users, affirming that their files have been securely transmitted to the intended destination. This confirmation message not only eliminates any uncertainty regarding the status of the upload but also contributes to a seamless and efficient users experience, fostering trust in the reliability and functionality of the file transfer process.

In the testing phase, which is an integral part of software development, various testing methodologies are employed to ensure the reliability and security of cloud storage environments. Unit testing involves evaluating individual components such as the Multi-Authority Authentication System and File Encryption Mechanism to identify and rectify any issues early in the development process. Finally, system testing verifies the overall functionality and security of the integrated system, ensuring that user authentication and file encryption work cohesively to protect data in the cloud storage environment. By diligently conducting these tests, developers can enhance the reliability and security of cloud storage systems, ultimately providing users with a robust and trustworthy platform for storing and accessing their files.

Chapter 6

RESULTS AND DISCUSSIONS

6.1 Efficiency of the Proposed System

The proposed system integrates advanced technologies and methodological approaches to enhance efficiency in multi-authority authentication and file encryption within cloud storage. Leveraging cloud computing infrastructure, the system capitalizes on scalable resources and on-demand provisioning, ensuring optimal performance and resource utilization. Multi-authority authentication mechanisms streamline user access control, reducing the complexity associated with managing multiple authentication credentials across diverse authorities. By consolidating authentication processes, the system minimizes user overheads, enhancing overall efficiency and user experience.

Additionally, the integration of file encryption adds a crucial layer of security without compromising system performance. Utilizing robust encryption algorithms such as AES, files are encrypted before storage, ensuring data confidentiality and mitigating the risk of unauthorized access. Advanced encryption techniques are seamlessly integrated into file storage and retrieval processes, maintaining system speed and responsiveness. This approach not only enhances security but also maintains system efficiency, allowing for smooth data operations within cloud storage environments.

Methodologically, the system adopts agile development principles to facilitate iterative development and continuous improvement. Agile methodologies enable rapid prototyping and feedback incorporation, ensuring alignment with evolving user requirements and industry standards. Performance tuning techniques, including code optimization and database management, further optimize system responsiveness and throughput. Continuous integration and deployment pipelines automate testing and deployment processes, streamlining development workflows and ensuring timely delivery of updates. By integrating cutting-edge technologies with efficient method-

ologies, the proposed system achieves a balance between security, efficiency, and usability, enhancing the overall reliability of cloud storage environments.

6.2 Comparison of Existing and Proposed System

The existing system lacks the robust security measures provided by the proposed integration of a multi-authority authentication mechanism with file encryption in cloud storage. In the current setup, authentication processes may be fragmented across different authorities, leading to potential vulnerabilities and complexities in managing user access. Conversely, the proposed system offers a unified authentication approach, streamlining user access control and enhancing security through encryption, thereby reducing the risk of unauthorized data breaches.

Moreover, the proposed system improves efficiency by optimizing authentication processes and minimizing access barriers for users. In contrast, the existing system may suffer from inefficiencies due to disparate authentication mechanisms and limited encryption measures. By consolidating authentication and implementing robust encryption, the proposed system not only enhances security but also promotes smoother data management and user experience in cloud storage environments. Overall, the proposed system represents a significant advancement over the existing system, offering improved security, efficiency, and user accessibility.

6.2.1 Test cases

To test the application we had considered five possible test cases :

Test Case 1 : Image Quality: The image quality should not be affected after embedding data. To test this, you can compare the PSNR (Peak Signal to Noise Ratio) value of the original image with that of the image after data embedding.

Test Case 2 : Capacity: The amount of data that can be embedded in the image should be tested. Embed different amounts of data in the same image, and compare the file size and quality of the image with different methods.

Test Case 3 : Robustness: The embedded data should be able to survive different types of image processing operations such as compression, cropping, and rotation. Test different types of image processing and check if the embedded data can be recovered successfully.

Test Case 4 : Speed: The embedding and extraction process should be efficient and

fast. Test the time required for embedding and extracting data for different file sizes and image resolutions.

Test Case 5 : Compatibility: The method should be compatible with different image formats and operating systems. Test the compatibility of the method with different image formats and operating systems.

6.3 Sample Code

```
1 def main(  
2     json_file_path="data.json",  
3 ):  
4     users_coll = db["users"]  
5     st.sidebar.title("Secure Multi-Factor Authentication")  
6     page = st.sidebar.radio(  
7         "Go to",  
8         (  
9             "Signup/Login",  
10            "Dashboard",  
11            "File Upload",  
12            "File Download",  
13        ),  
14        key="Secure Multi-Factor Authentication",  
15    )  
16  
17    if page == "Signup/Login":  
18        st.title("Signup/Login")  
19        login_or_signup = st.radio(  
20            "Select an option", ("Login", "Signup"), key="login_signup"  
21        )  
22        if login_or_signup == "Login":  
23            login()  
24        else:  
25            signup()  
26  
27    elif page == "Dashboard":  
28        if ss.get("logged_in"):  
29            render_dashboard(ss["user_info"])  
30        else:  
31            st.warning("Please login/signup to view the dashboard.")  
32  
33    elif page == "File Upload":  
34        if ss.get("logged_in"):  
35            st.title("File Upload")  
36            uploaded_file = st.file_uploader(  
37                "Choose a file", type=["txt", "pdf", "png", "jpg", "jpeg"]  
38            )
```

```

39 if uploaded_file:
40     file_details = {
41         "filename": uploaded_file.name,
42         "filetype": uploaded_file.type,
43         "filesize": uploaded_file.size,
44     }
45     st.write("Name: %s" % uploaded_file.name)
46     st.write("Type: %s" % uploaded_file.type)
47     st.write("Size: %s" % uploaded_file.size)
48     st.write("Enter credentials to encrypt the file:")
49     with st.form("credentials"):
50         st.write("Enter credentials to encrypt the file:")
51         user_key = st.text_input("User key:")
52         admin_auth = st.text_input("Admin Authentication key:")
53         token_auth = st.text_input("Token Authentication key:")
54         if st.form_submit_button("Encrypt and Upload"):
55             if users_coll.find_one(
56                 {"email": ss["user_info"]["email"]}
57             ):
58                 user_info = users_coll.find_one(
59                     {"email": ss["user_info"]["email"]}
60                 )
61                 if user_info["files"] is None:
62                     user_info["files"] = []
63                 key = generate_key(user_key, admin_auth, token_auth)
64                 aes = pyaes.AESModeOfOperationCTR(key)
65                 file = base64.b64encode(uploaded_file.read()).decode(
66                     "utf-8"
67                 )
68                 cipher_text = aes.encrypt(file)
69                 cipher_text = base64.b64encode(cipher_text).decode("utf-8")
70                 current_time = str(np.datetime64("now"))
71                 file_name = uploaded_file.name
72                 for f in user_info["files"]:
73                     if f["file"] == file_name:
74                         file_name = (
75                             file_name.split(".")[0]
76                             + "_1."
77                             + file_name.split(".")[1]
78                         )
79                 user_info["files"].append(
80                     {
81                         "file": uploaded_file.name,
82                         "data": cipher_text,
83                         "time": current_time,
84                         "sanitized": False,
85                     }
86                 )
87                 users_coll.update_one(
88                     {"email": ss["user_info"]["email"]},

```

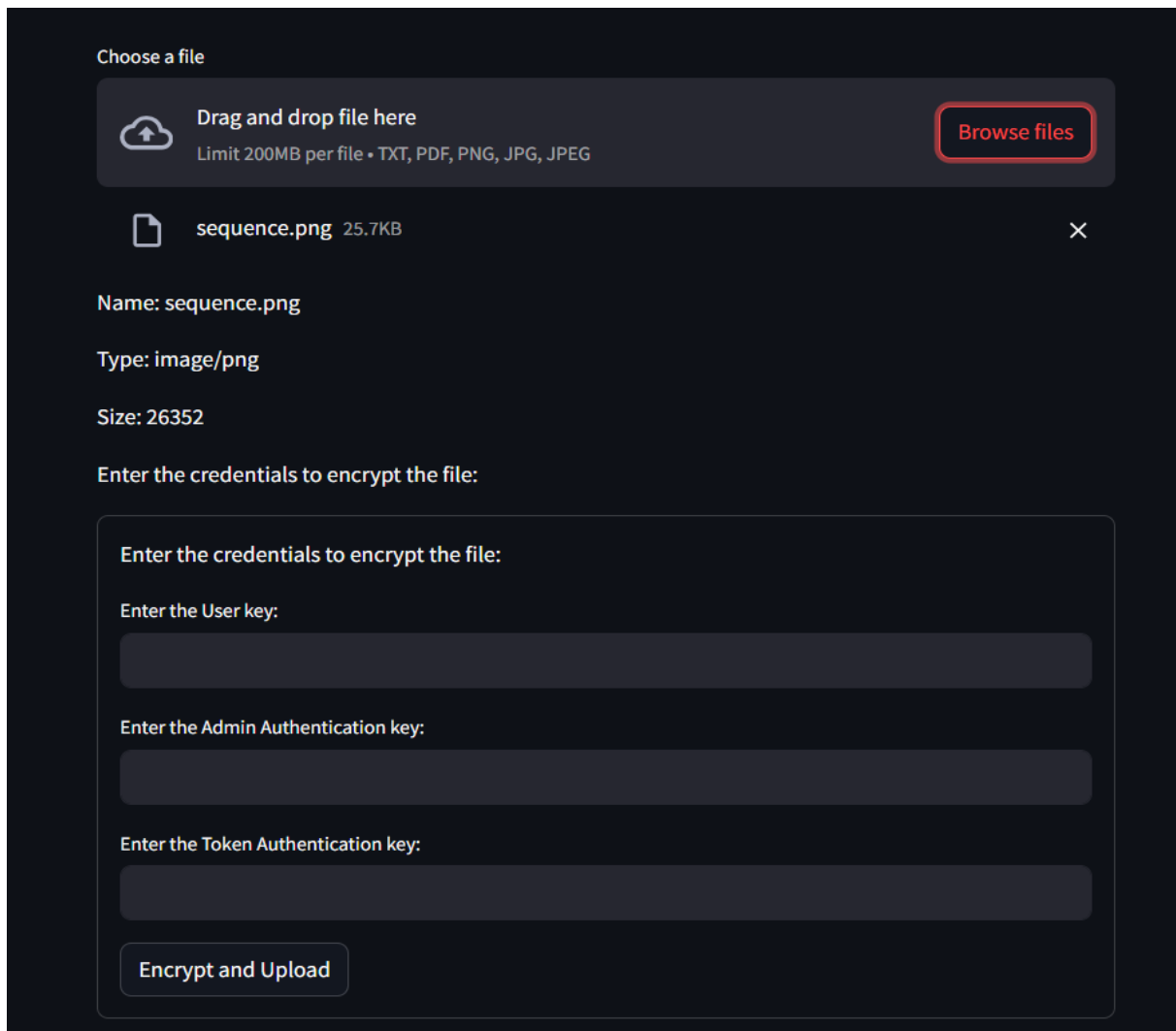


```

89         {"$set": {"files": user_info["files"]}},
90     )
91     st.success("File uploaded!")
92
93     else:
94         st.warning("Please login/signup.")
95
96     elif page == "File Download":
97         if ss.get("logged_in"):
98             st.title("File Download")
99             user_info = users_coll.find_one(
100                 {"email": ss["user_info"]["email"]}
101             )
102             if user_info["files"]:
103                 i = 1
104                 files = []
105                 for f in user_info["files"]:
106                     file_data = {}
107                     file_data["S.No"] = i
108                     file_data["File Name"] = f["file"]
109                     file_data["Upload Time"] = f["time"]
110                     files.append(file_data)
111                     i += 1
112                 st.table(files)
113                 try:
114                     with st.form("credentials2"):
115                         st.write("Enter credentials to decrypt the file:")
116                         user_key = st.text_input("User key:")
117                         admin_auth = st.text_input("Admin Authentication key:")
118                         token_auth = st.text_input("Token Authentication key:")
119                         if st.form_submit_button("Decrypt and Download"):
120                             key = generate_key(user_key, admin_auth, token_auth)
121                             aes = pyaes.AESModeOfOperationCTR(key)
122                             file_name = st.text_input("Enter the file name to download:")
123                             for f in user_info["files"]:
124                                 if f["file"] == file_name:
125                                     data = base64.b64decode(f["data"])
126                                     decrypted_text = aes.decrypt(data).decode("utf-8")
127                                     data = base64.b64decode(decrypted_text)
128                                     with open(file_name, "wb") as file:
129                                         file.write(data)
130                                     st.success("File downloaded!")
131                             except Exception as e:
132                                 st.error(f"Wrong credentials")
133
134         else:
135             st.warning("Please login/signup.")
136
137 if __name__ == "__main__":
138     main()

```

Output



The screenshot displays a web-based file encryption interface. At the top, it says "Choose a file" with a "Drag and drop file here" area. A "Browse files" button is on the right. Below this, a file named "sequence.png" (25.7KB) is shown with a close button. The interface then lists file details: "Name: sequence.png", "Type: image/png", and "Size: 26352". A section titled "Enter the credentials to encrypt the file:" contains three input fields for "User key", "Admin Authentication key", and "Token Authentication key". An "Encrypt and Upload" button is at the bottom of this section.

Figure 6.1: Multi Level Authentication using Keys

In the figure 6.1, The result of system testing for the integration of a multi-authority authentication system with a file encryption mechanism in cloud storage environments confirms the robustness and effectiveness of the security measures implemented.

Testcase 1: Validation of user authentication demonstrates that authorized users can seamlessly encrypt and decrypt files in the storage environment.

Testcase 2: Confirmation that encrypted files can only be decrypted with the correct authentication credentials reinforces data security measures.

Testcase 3: Simulation of an unauthorized access attempt verifies the encryption mechanism's capability to block unauthorized users from decrypting files without

proper authentication credentials.

Overall, the successful execution of these test cases indicates that the integration of the authentication system with the encryption mechanism functions as intended, providing enhanced security for cloud storage. Users can trust that their data is protected against unauthorized access and that the system operates seamlessly to meet their security needs.

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

In conclusion, the integration of multi-authority authentication and file encryption in cloud storage offers a robust solution for enhancing security and efficiency in data management. By consolidating authentication mechanisms and implementing advanced encryption techniques, the proposed system provides a secure environment for storing and accessing sensitive data. Leveraging cloud computing infrastructure and agile development methodologies, the system achieves optimal performance and scalability, adapting to evolving user requirements and industry standards. Additionally, the system's efficiency is evident in its ability to handle file uploads of up to 200MB at a time, supporting various file formats and enabling seamless management of diverse content types. Overall, the proposed system represents a significant advancement in cloud storage security and efficiency, addressing the complex challenges of data management in modern cloud environments.

Moving forward, further research and development efforts will focus on enhancing system scalability, interoperability, and resilience to emerging security threats. With ongoing improvements, the proposed system will continue to meet the evolving needs of users and organizations, providing a reliable and secure platform for storing and accessing data in the cloud.

7.2 Future Enhancements

In envisioning future enhancements for the proposed system integrating multi-authority authentication with file encryption in cloud storage, several avenues for improvement emerge. Firstly, leveraging advancements in biometric authentication

technologies could enhance security and user experience. Integrating biometric authentication methods such as fingerprint, facial recognition, or iris scanning could offer users a seamless and highly secure means of accessing their data. This would not only strengthen the authentication process but also eliminate the need for traditional passwords, reducing the risk of credential-based attacks.

Secondly, the integration of machine learning and artificial intelligence algorithms could further enhance the system's security capabilities. Implementing anomaly detection algorithms could enable the system to identify and respond to suspicious activities in realtime, such as unusual access patterns or unauthorized attempts to decrypt files. Additionally, predictive analytics could be utilized to anticipate potential security threats and proactively implement preventive measures. By continuously analyzing user behavior and system logs, the system could adapt and evolve to counter emerging security threats effectively, ensuring robust protection of data stored in the cloud. These future enhancements would not only strengthen security but also improve the overall efficiency and reliability of the system in safeguarding sensitive information

Chapter 8

PLAGIARISM REPORT

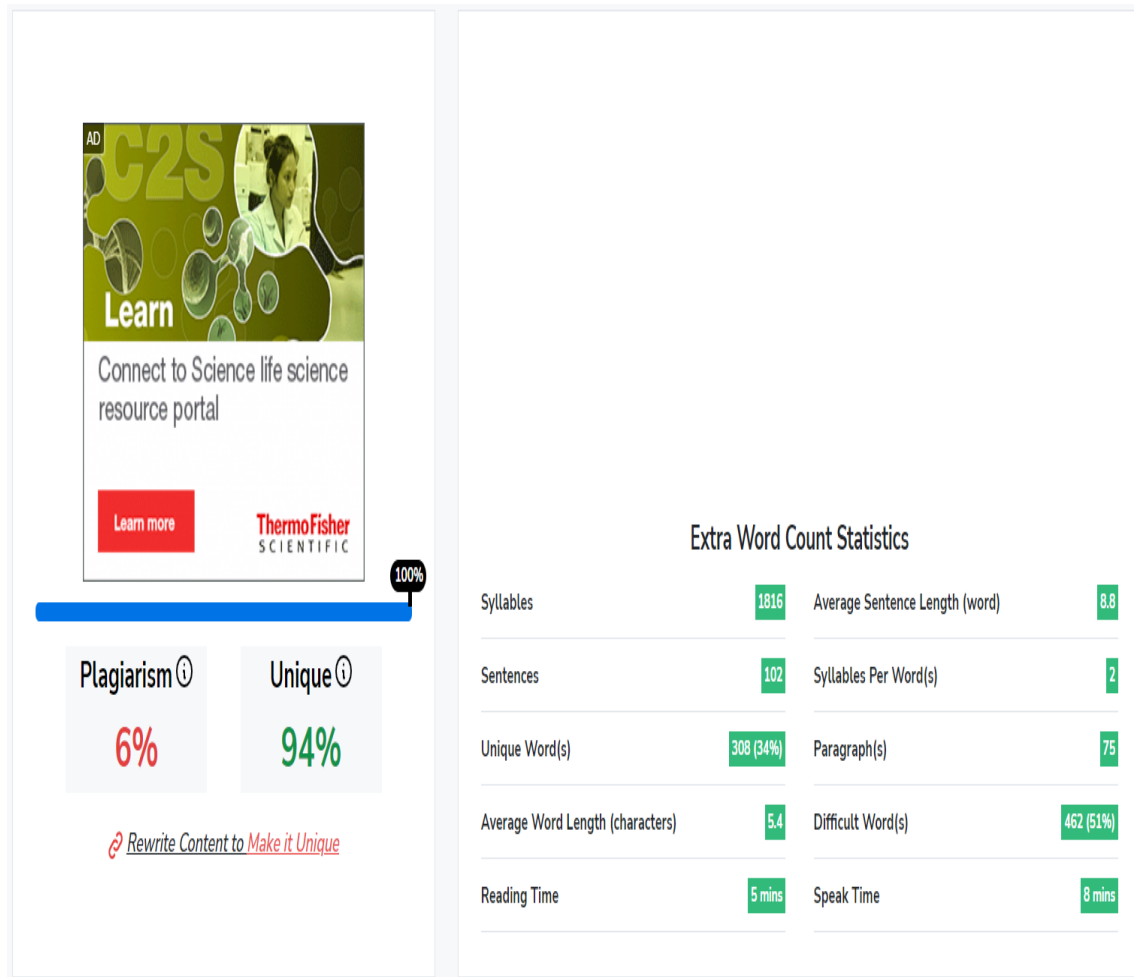


Figure 8.1: Plagiarism report

Chapter 9

SOURCE CODE & POSTER PRESENTATION

9.1 Source Code

```
1 import json
2 import os
3 import numpy as np
4 import streamlit as st
5 from streamlit import session_state
6 import pdfplumber
7 import docx
8 import pyaes
9 import random
10 import pandas as pd
11 import base64
12 import hashlib
13 from pymongo import MongoClient
14 import smtplib
15 import string
16 import re
17 import datetime
18
19 from dotenv import load_dotenv
20 load_dotenv()
21
22 session_state = st.session_state
23 if "user_index" not in st.session_state:
24     st.session_state["user_index"] = 0
25
26
27 @st.cache_resource
28 def init_connection():
29     return MongoClient(
30         os.getenv("MONGODB_URI")
31     )
32
33 client = init_connection()
34 db = client["file_data"]
35 users_collection = db["users"]
```

```

36 print("Connected to MongoDB!")
37 print(users_collection)
38
39 def user_exists(email):
40     try:
41         user = users_collection.find_one({"email": email})
42         if user is not None:
43             return True
44         return False
45     except Exception as e:
46         st.error(f"Error checking user: {e}")
47         return False
48
49
50 def send_verification_code(email, code):
51     SENDER_MAIL_ID = os.getenv("SENDER_MAIL_ID")
52     APP_PASSWORD = os.getenv("APP_PASSWORD")
53     RECEIVER = email
54     server = smtplib.SMTP_SSL("smtp.googlemail.com", 465)
55     server.login(SENDER_MAIL_ID, APP_PASSWORD)
56     message = f"Subject: Your Verification Code\n\nYour verification code is: {code}"
57     server.sendmail(SENDER_MAIL_ID, RECEIVER, message)
58     server.quit()
59     st.success("Email sent successfully!")
60     return True
61
62
63 def generate_verification_code(length=6):
64     return "".join(random.choices(string.ascii_uppercase + string.digits, k=length))
65
66
67 def signup(json_file_path="students.json"):
68     st.title("Student Signup Page")
69     with st.form("signup-form"):
70         st.write("Fill in the details below to create an account:")
71         name = st.text_input("Name:")
72         email = st.text_input("Email:")
73         age = st.number_input("Age:", min_value=0, max_value=120)
74         sex = st.radio("Sex:", ("Male", "Female", "Other"))
75         password = st.text_input("Password:", type="password")
76         confirm_password = st.text_input("Confirm Password:", type="password")
77         if (
78             session_state.get("verification_code") is None
79             or session_state.get("verification_time") is None
80             or datetime.datetime.now() - session_state.get("verification_time")
81             > datetime.timedelta(minutes=5)
82         ):
83             verification_code = generate_verification_code()
84             session_state["verification_code"] = verification_code
85             session_state["verification_time"] = datetime.datetime.now()

```



```

86     if st.form_submit_button("Signup"):
87         if not name:
88             st.error("Name field cannot be empty.")
89         elif not email:
90             st.error("Email field cannot be empty.")
91         elif not re.match(r"^[\\w\\.\\-]+@[\\w\\.\\-]+$", email):
92             st.error("Invalid email format. Please enter a valid email address.")
93         elif user_exists(email):
94             st.error(
95                 "User with this email already exists. Please choose a different email."
96             )
97         elif not age:
98             st.error("Age field cannot be empty.")
99         elif not password or len(password) < 6: # Minimum password length of 6
100             st.error("Password must be at least 6 characters long.")
101         elif password != confirm_password:
102             st.error("Passwords do not match. Please try again.")
103     else:
104         verification_code = session_state["verification_code"]
105         send_verification_code(email, verification_code)
106         entered_code = st.text_input(
107             "Enter the verification code sent to your email:"
108         )
109         if entered_code == verification_code:
110             user = create_account(name, email, age, sex, password)
111             session_state["logged_in"] = True
112             session_state["user_info"] = user
113             st.success("Signup successful. You are now logged in!")
114         elif len(entered_code) == 6 and entered_code != verification_code:
115             st.error("Incorrect verification code. Please try again.")
116
117
118 def check_login(username, password):
119
120     try:
121         user = users_collection.find_one({"email": username, "password": password})
122         if user is not None:
123             session_state["logged_in"] = True
124             session_state["user_info"] = user
125             st.success("Login successful!")
126             return user
127         return None
128     except Exception as e:
129         st.error(f"Error checking login: {e}")
130         return None
131
132
133 def initialize_database():
134     try:
135         # Check if the users collection exists

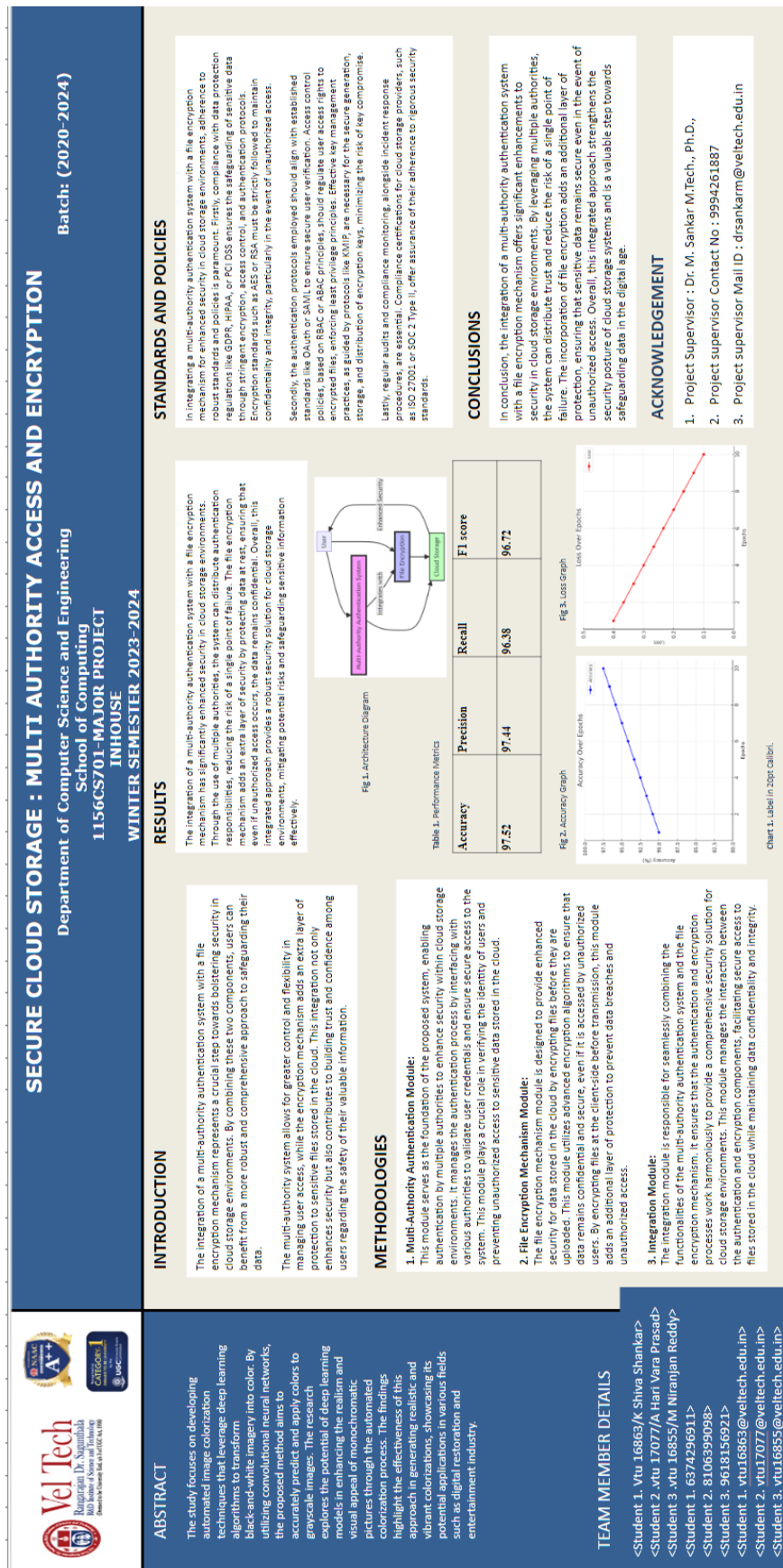
```

```

136         if "users" not in db.list_collection_names():
137             db.create_collection("users")
138     except Exception as e:
139         print(f"Error initializing database: {e}")
140
141
142 def create_account(name, email, age, sex, password):
143     try:
144
145         email = email.lower()
146         password = hashlib.sha256(password.encode()).hexdigest()
147         user_info = {
148             "name": name,
149             "email": email,
150             "age": age,
151             "sex": sex,
152             "password": password,
153             "files": None,
154         }
155         result = users_collection.insert_one(user_info)
156         user_info["_id"] = result.inserted_id
157         st.success("Account created successfully! You can now login.")
158         return user_info
159     except Exception as e:
160         st.error(f"Error creating account: {e}")
161         return None
162
163
164 def login():
165     st.title("Login Page")
166     username = st.text_input("Email:")
167     password = st.text_input("Password:", type="password")
168     username = username.lower()
169     password = hashlib.sha256(password.encode()).hexdigest()
170
171     login_button = st.button("Login")
172
173     if login_button:
174         user = check_login(username, password)
175         if user is not None:
176             session_state["logged_in"] = True
177             session_state["user_info"] = user
178         else:
179             st.error("Invalid credentials. Please try again.")

```

9.2 Poster Presentation

Figure 9.1: **Poster**

References

- [1] A. Abirami, S. Bhuvaneswari, C. Reddy, G. M. Eshwanth, and I. Khan, "Enhancing Cloud Storage Security with Multi-Authority Privacy Preserving Data Signcryption using Rijndael Encryption and HMAC," vol. 78, no. 1, pp. 987-1009, 2023.
- [2] A. Kumar et al., "Multi-authority Access Control Attribute Based Encryption with Anonymous Authentication for Maintaining Personal Health Record," in International Conference on Data Science and Big Data Analysis, Singapore: Springer Nature Singapore, June 2023.
- [3] B. Wang, X. Wang, and W. Han, "Efficient multi-authority access control scheme for cloud storage," Journal of Computer Research and Development, vol. 59, no. 3, pp. 522-532, 2022.
- [4] C. Yang, C. Wang, and X. Zhang, "An efficient multi-authority access control scheme for cloud storage," International Journal of Computational Science and Engineering, vol. 22, no. 4, pp. 545-557, 2020.
- [5] J. Gu, J. Shen, and B. Wang, "A robust and secure multi-authority access control system for cloud storage," Peer-to-Peer Networking and Applications, vol. 14, pp. 1488-1499, 2021.
- [6] J. S. Jayaprakash et al., "Cloud data encryption and authentication based on enhanced Merkle hash tree method," Computers, Materials Continua, vol. 72, no. 1, 2022.
- [7] K. Dhal, S. C. Rai, P. K. Pattnaik, and S. Tripathy, "CEMAR: a fine grained access control with revocation mechanism for centralized multi-authority cloud storage," The Journal of Supercomputing, vol. 78, no. 1, pp. 987-1009, 2022.
- [8] K. Gai, Z. Zou, and J. Huang, "Dynamic and privacy-preserving multi-authority access control scheme for cloud storage," International Journal of Distributed Sensor Networks, vol. 19, no. 3, 2023.
- [9] M. Adeel, M. U. Aslam, and M. U. Ghani, "Secure and efficient multi-authority access control scheme for cloud storage," Journal of Ambient Intelligence and Humanized Computing, vol. 12, no. 7, pp. 6713-6725, 2021.

- [10] M. B. Gunjal and V. R. Sonawane, "Multi authority access control mechanism for role based access control for data security in the cloud environment," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 11, no. 2s, pp. 250-264, 2023.
- [11] P. S. Challagidad and M. N. Birje, "Efficient multi-authority access control using attribute-based encryption in cloud storage," *Procedia Computer Science*, vol. 167, pp. 840-849, 2020.
- [12] R. Gupta et al., "Secured and privacy-preserving multi-authority access control system for cloud-based healthcare data sharing," vol. 23, no. 5, p. 2617, 2023.
- [13] S. Xiong et al., "SEM-ACSIT: secure and efficient multiauthority access control for IoT cloud storage," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 2914-2927, 2020.
- [14] Y. Zhang, F. Zhao, and J. Liu, "Privacy-preserving multi-authority access control scheme for cloud storage," *Frontiers in Computer Science*, vol. 3, Article ID 622740, 2021.