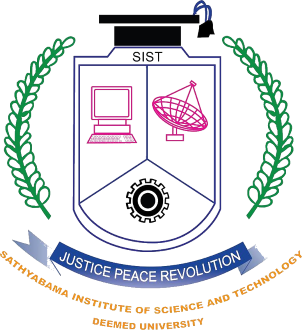
# AGILE CLOUD ENCRYPTION FRAMEWORK

Submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering

by

## CHINNAM ESWAR (Reg. No - 41110270)

## PEDASINGU JAYA PRAKASH (Reg. No - 41110943)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# SCHOOL OF COMPUTING

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY (DEEMED TO BE UNIVERSITY)**

**CATEGORY -1 UNIVERSITY BY UGC**

**Accredited with Grade “A++” by NAAC I 12B Status by UGC I Approved by AICTE**

**JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119**

# APRIL - 2025



## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **CHINNAM ESWAR (41110270) and PEDASINGU JAYA PRAKASH (41110943)** who carried out the Project entitled **“AGILE CLOUD ENCRYPTION FRAMEWORK”** under my supervision from November 2024 to April 2025.

## Internal Guide

**Ms. E.VINOTHINI, M.E, Assistant Professor, CSE**

## Head of the Department

**Dr. L. LAKSHMANAN, M.E., Ph.D.**

## Submitted for Viva voce Examination held on

**Internal Examiner External Examiner**

ii

# DECLARATION

I, **CHINNAM ESWAR**  **(Reg. No - 41110270),** hereby declare that the Project Report entitled **“AGILE CLOUD ENCRYPTION FRAMEWORK”** done by me under the guidance of **Ms. E.VINOTHINI, M.E, Assistant Professor, CSE** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

## DATE :

**PLACE : Chennai SIGNATURE OF THE CANDIDATE**

# ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of** **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph. D**, **Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.,** Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide is **Ms. E.VINOTHINI, M. E, Assistant Professor, CSE.** for her valuable guidance and the suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

# ABSTRACT

The implementation of an encrypted file chunking system relies on the utilization of cloud infrastructure and hashing algorithms to ensure the secure storage and transmission of data. This system aims to enhance data protection by breaking files into smaller chunks and encrypting them individually before storing them in the cloud. The use of cloud infrastructure provides scalability, ease of access, and cost- effectiveness, making it an ideal choice for storing large volumes of data securely. Hashing algorithms play a crucial role in this system by generating unique hash values for each file chunk, which are used for integrity verification. These algorithms ensure that the data remains intact during storage and transmission by detecting any unauthorized modifications. By combining encryption, chunking, and hashing, this implementation provides a comprehensive solution for protecting sensitive information in the cloud. It not only safeguards data against unauthorized access but also ensures its integrity, making it suitable for various applications such as secure file sharing and backup services. Additionally, this system can be further enhanced by implementing additional security measures such as access controls and data redundancy. The integration of encrypted file chunking with cloud infrastructure and hashing algorithms offers a robust foundation for implementing secure and efficient data storage and transmission systems.

**Keywords:** PERFORMANCE, FRAGMENTATION, REPLICATION, CHUNKING CLOUD SECURITY AND CENTRALIZATION.

# TABLE OF CONTENTS

|  |  |  |  |
| --- | --- | --- | --- |
| **Chapter No** |  | **TITLE** | **Page No.** |
|  | **ABSTRACT** | | v |
|  | **LIST OF FIGURES** | | VIII |
|  | **LIST OF ABBREVIATIONS** | | IX |
| 1 | **INTRODUCTION** | | 1 - 4 |
|  | 1.1 Introduction | | 1 |
|  | 1.2 Overview of file encryption and chunking system | | 3 |
|  | 1.3 Importance of Data security in clod computing | | 3 |
|  | 1.4 Development Process of the Encryption system | | 4 |
| 2 | **LITERATURE SURVEY** | | 5 - 11 |
|  | 2.1 Inferences from Literature Survey | | 5 |
|  | 2.2 Existing System and Proposed System | | 8 |
| 3 | **AIM AND SCOPE OF THE PRESENT INVESTIGATION** | | 12 - 16 |
|  | 3.1 | Aim and Scope | 12 |
|  | 3.2 | Requirement Specifications | 16 |
| 4 | **EXPERIMENTAL OR MATERIALS AND METHODS ALGORITHM USED** | | 17 - 25 |
|  | 4.1 | System Architecture | 17 |
|  | 4.2 | Flow Chart | 18 |
|  | 4.3 | Data Encryption Techniques | 19 |
|  | 4.4 | Model Improvisation | 21 |
|  | 4.5 | Graphs | 23 |
|  | 4.6 | Creating User Interface | 24 |
| 5 | **RESULT AND DISCUSSION, PERFORMANCE ANALYSIS** | | 26 - 27 |
| 6 | **SUMMARY AND CONCLUSION** | | 28 |
|  |  | |  |
|  | **REFERENCES** | | 29 |
|  | **APPENDIX** | | 31 |
|  | **1. SOURCE CODE** | | 31 |
|  | **2. OUTPUT SCREENSHOTS** | | 62 |
|  |  | |  |
|  |  | |  |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO** | **FIGURE NAME** | **Page No.** |
| 1.1.1 | IMPLEMTATION DIAGRAM | 2 | |
| 2.2.1 | MULTI-LAYER ENCRYPTION | 9 | |
| 4.1.1 | SYSTEM ARCHITECTURE | 17 | |
| 4.2.1 | FLOW CHART | 18 | |

4.5.1 TOTAL NUMBER OF DATA BREACHES AND TEAKS 23

4.5.2 COMPARISION BETWEEN DIFFERENT ALGORITHMS 23

4.5.3 DATA BREACH AND CYBER ATTACKS IN DIFFERENT 24

ZONES

A 2.1 HOME PAGE 57

A 2.2 LOGIN PAGE 57

A 2.3 SIGNUP PAGE 58

A 2.4 AUTHENTICATION PAGE 58

A 2.5 FILE UPLOADE PAGE 59

A 2.6 MY FILE MANAGEMENT 59

A 2.7 USER CREATION DATA 60

A 2.7 FILES CHUNKED DATA 60

# LIST OF ABBREVIATIONS

|  |  |
| --- | --- |
| **ABBREVIATIONS** | **EXPANSION** |
| AES | Advance Encryption Standard |
| RSA | Rivest-Shamir-Adleman |
| OTP | One Time Password |
| SHA | Secure Hash Algorithm |

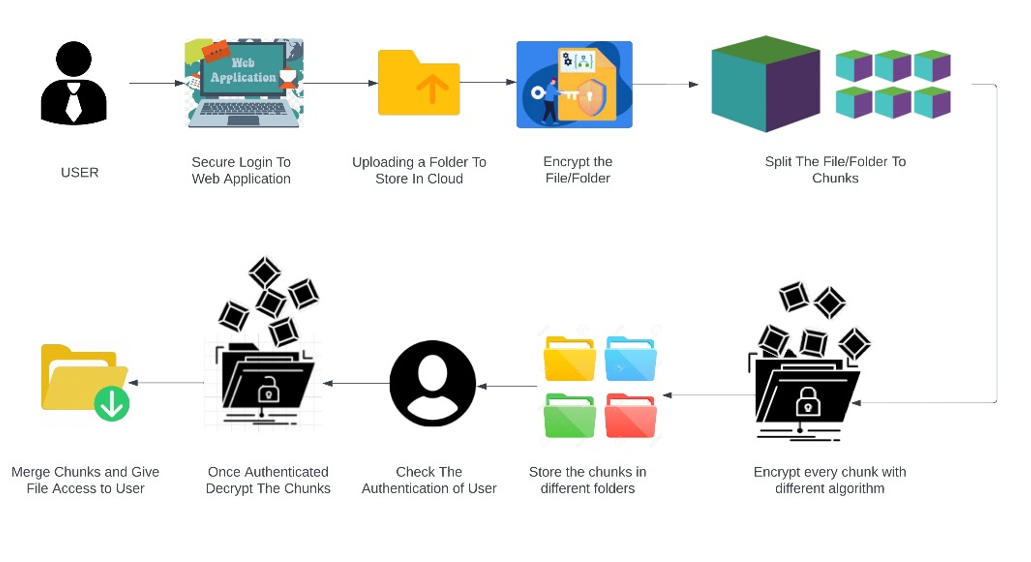
**CHAPTER 1 INTRODUCTION**

## 1.1 INTRODUCTION

The implementation of an encrypted file chunking system using cloud infrastructure and hashing algorithms aims to enhance the security and efficiency of data storage and retrieval processes. This system would revolutionize the way files are stored and accessed, ensuring utmost protection against unauthorized access and data breaches. By implementing this system, organizations can ensure the confidentiality, integrity, and availability of their valuable data assets.

Cloud infrastructure plays a crucial role in this system as it provides scalability, flexibility, and cost-efficiency. Cloud service providers offer vast amounts of storage space, enabling organizations to securely store large volumes of data without worrying about hardware limitations. Additionally, the use of cloud infrastructure allows for easy scaling up or down based on the organization's storage needs. This ensures that companies only pay for the storage they use, reducing unnecessary expenses. The file chunking aspect of the system breaks large files into smaller, more manageable chunks. This division promotes efficient storage and retrieval as smaller chunks can be processed and transferred more quickly than a single large file. Moreover, in the event of data corruption or loss, only the affected chunk needs to be retransmitted, minimizing the impact on overall system performance. To ensure data security during storage and transmission, encryption is applied to each chunk. Encryption scrambles the data using cryptographic algorithms, rendering it unreadable without the corresponding decryption keys. By doing so, even if data is intercepted during transmission or if unauthorized access is gained, the encrypted chunks remain incomprehensible, safeguarding sensitive information. Alongside encryption, hashing algorithms play a crucial role in this system. Each file chunk is assigned a unique hash value, which acts as a digital fingerprint. Hashing algorithms enable secure content identification and integrity checking. Any modification or tampering of a file chunk will result in a different hash value, alerting the system of potential data corruption or unauthorized access. This ensures the integrity of the stored data, providing organizations with confidence in the accuracy and reliability of their files. The implementation of this system offers several significant benefits. Firstly, it enhances the security of the organization's data assets. By encrypting each file chunk and utilizing hashing algorithms for integrity checks, the system significantly reduces the risk of unauthorized access, data tampering, or data breaches. Companies can be assured that their sensitive information remains confidential and that data integrity is maintained throughout the storage and transmission process.

# IMPLEMENTATION DIAGRAM

***Fig 1.1.1 :* Implementation Diagram**

Secondly, it improves the efficiency of storing and retrieving data. By breaking large files into smaller, more manageable chunks, the system optimizes storage and transmission processes, reducing the overall time required for these operations. This enhances the system's scalability, allowing organizations to handle increased data volumes without compromising performance. Lastly, the use of cloud infrastructure offers cost benefits and flexibility. Utilizing cloud storage eliminates the need for expensive on-premises hardware or infrastructure investments. Instead, organizations can leverage the cloud provider's infrastructure, paying only for the storage and computing resources they consume. This allows companies to scale their storage needs up or down as required, adapting to changing business demands without incurring unnecessary costs. In conclusion, the implementation of an encrypted file chunking system using cloud infrastructure and hashing algorithms offers a secure and efficient solution for data storage and retrieval processes. By leveraging cloud infrastructure, organizations can benefit from scalability, flexibility, and cost-efficiency. Encryption and hashing algorithms enhance data security, ensuring data confidentiality and integrity throughout the storage and transmission process. Implementing this system equips organizations with the tools needed to protect their valuable data assets, providing peace of mind and enabling them to focus on their core business operations.

## OVERVIEW OF FILE ENCRYPTION AND CHUNKING SYSTEM

File encryption and chunking systems are critical components in enhancing data security and management within a cloud computing infrastructure. These systems work by encrypting the file data using strong cryptographic algorithms to render the information unreadable without the proper decryption keys. By breaking down the file into smaller chunks, the chunking system enables more efficient storage, transmission, and processing of data in the cloud. This approach not only enhances security by dispersing the file across multiple chunks but also improves data management by allowing for easier access and manipulation of specific parts of the file. Implementing file encryption and chunking systems within a cloud environment significantly reduces the risk of unauthorized access or data breaches, ensuring that sensitive information remains protected at all times. Additionally, the use of these systems provides a practical solution for securely storing and sharing files across different cloud platforms, ultimately enhancing overall data security and management capabilities within the cloud computing infrastructure.

**1.3 IMPORTANCE OF DATA SECURITY IN CLOUD COMPUTING**

Data security is paramount in the context of cloud computing, particularly when developing and implementing a file encryption and chunking system within a cloud infrastructure. Firstly, ensuring data security in cloud computing environments helps in protecting sensitive information from unauthorized access, data breaches, and cyber attacks. By implementing robust encryption techniques and secure chunking mechanisms, organizations can safeguard their data against potential threats both during transmission and storage. Additionally, maintaining data security also helps in meeting regulatory compliance requirements, thus avoiding legal repercussions and financial penalties. Furthermore, enhancing data security through encryption and chunking contributes to building trust and confidence among users, clients, and stakeholders, promoting a positive reputation for the organization. Overall, prioritizing data security in cloud computing infrastructure is crucial for safeguarding confidential information, mitigating risks, and ensuring smooth operations within an increasingly digital and interconnected world.

**1.4 DEVELOPMENT PROCESS OF THE ENCRYPTION SYSTEM**

The development process of the encryption system for a file encryption and chunking system within a cloud computing infrastructure involves several key steps to enhance data security and management. Firstly, it is essential to conduct thorough research on encryption algorithms and techniques to select the most suitable approach for the system. This includes understanding the requirements of the system, assessing the level of security needed, and considering factors such as performance and compatibility. Secondly, the encryption system's architecture must be designed, implementing cryptographic algorithms for file encryption and chunking mechanisms to break down the data into smaller parts for enhanced security. This step also involves integrating key management processes to securely store and manage encryption keys. Thirdly, rigorous testing and evaluation of the encryption system are vital to ensure its effectiveness and identify any vulnerabilities or weaknesses that could be exploited. This testing phase includes both functional and security testing to validate the system's performance and resilience against potential attacks. Throughout the development process, close collaboration between developers, security experts, and system administrators is crucial to address any challenges and ensure the successful implementation of the encryption system within the cloud computing infrastructure for robust data security and management.

# CHAPTER 2 LITERATURE SURVEY

## INFERENCES FROM LITERATURE SURVEY

**[1]. Seth, B., Dalal, S., Jaglan, V., Le, D. N., Mohan, S., & Srivastava, G. (2022). Integrating encryption techniques for secure data storage in the cloud. Transactions on Emerging Telecommunications Technologies, 33(4), e4108.**

Seth et al. (2022) presented a study focused on integrating encryption techniques for secure data storage in the cloud. The research, published in Transactions on Emerging Telecommunications Technologies, aimed to enhance data security and management within a cloud computing infrastructure. The authors developed and implemented a File Encryption and Chunking System as part of their approach, emphasizing the importance of secure data handling in cloud environments. This work contributes to advancing methods for safeguarding sensitive information stored in the cloud.

[**2]. Seth, B., Dalal, S., Le, D. N., Jaglan, V., Dahiya, N., Agrawal, A., ... & Verma, K. D. (2021). Secure Cloud Data Storage System Using Hybrid Paillier–Blowfish Algorithm. Computers, Materials & Continua, 67(1).**

In their 2021 study, Seth et al. introduced a Secure Cloud Data Storage System utilizing the Hybrid Paillier–Blowfish Algorithm. The system aimed at enhancing data security and management within a cloud computing infrastructure. Their approach involved file encryption and chunking techniques for improved data protection. The research published in Computers, Materials & Continua highlighted the significance of utilizing advanced encryption algorithms for secure cloud storage applications.

[**3]. Akbar, M., Ahmad, I., Mirza, M., Ali, M., & Barmavatu, P. (2023). Enhanced authentication for de-duplication of big data on cloud storage system using machine learning approach. Cluster Computing, 1-20.**

Akbar, M., Ahmad, I., Mirza, M., Ali, M., & Barmavatu, P. (2023) conducted a study on enhanced authentication for de-duplication of big data on cloud storage systems using a machine learning approach. Their research, published in Cluster Computing, focused on improving data security and management through the development and implementation of a file encryption and chunking system within a cloud computing infrastructure. The study aimed to enhance data security and efficiency in cloud storage systems by leveraging machine learning techniques for de-duplication and authentication processes.

[**4]. Singh, R., Kumar Singh, A., Pandey, V., Sharma, M., & Sharma, A. (2022). Parallel Chunk Encryption in Public Cloud Storage: Validating Data Privacy. Meenakshi and Sharma, Akhilesh, Parallel Chunk Encryption in Public Cloud Storage: Validating Data Privacy (July 14, 2022).**

Singh, R., Kumar Singh, A., Pandey, V., Sharma, M., & Sharma, A. (2022) conducted a study on Parallel Chunk Encryption in Public Cloud Storage, focusing on validating data privacy. Their research emphasized the importance of data security and management within cloud computing infrastructure. Implementing a File Encryption and Chunking System, the authors aimed to enhance the overall data protection measures. This study contributes to advancements in data privacy practices in the digital age.

**[5]. Krishnamoorthy, N., & Umarani, S. (2023). Implementation and management of cloud security for industry 4. O-data using hybrid elliptical curve cryptography. The Journal of High Technology Management Research, 34(2), 100474.**

Krishnamoorthy, N., & Umarani, S. (2023) explore the implementation and management of cloud security for industry 4.0 data using hybrid elliptical curve cryptography in their article published in The Journal of High Technology Management Research. They delve into the development and implementation of a file encryption and chunking system within a cloud computing infrastructure, aiming to enhance data security and management. This research highlights the significance of employing advanced cryptographic techniques to safeguard data in the context of evolving industry standards and technological advancements.

**[6]. Narayanan, U., Paul, V., & Joseph, S. (2022). A novel system architecture for secure authentication and data sharing in cloud enabled Big Data Environment. Journal of King Saud University-Computer and Information Sciences, 34(6), 3121-3135.**

Narayanan, Paul, and Joseph (2022) proposed a novel system architecture for secure authentication and data sharing in a cloud-enabled Big Data environment. The research, published in the Journal of King Saud University-Computer and Information Sciences, focuses on developing a File Encryption and Chunking System within a cloud computing infrastructure to enhance data security and management. Through their innovative approach, they aim to address the challenges associated with secure authentication and data sharing in the context of Big Data, providing a valuable contribution to the field of cloud computing and data security.

**[7]. Adhab, A. H., & Hussien, N. A. (2023). Study of Efficient Cloud Storage Architectures for the Security Environment. Journal of Kufa for Mathematics and Computer, 10(1), 63-71.**

Adhab and Hussien (2023) conducted a study on efficient cloud storage architectures within the security environment, as published in the Journal of Kufa for Mathematics and Computer. Their research focused on exploring various strategies to enhance the security of data stored in the cloud, aiming to optimize storage systems while ensuring robust protection against potential threats. Through their investigation, they identified key considerations and proposed solutions to address security challenges in cloud storage environments. Their findings contribute valuable insights to the ongoing efforts in developing secure and efficient cloud storage solutions.

**[8]. Adee, R., & Mouratidis, H. (2022). A dynamic four-step data security model for data in cloud computing based on cryptography and steganography. Sensors, 22(3), 1109.**

Adee and Mouratidis (2022) proposed a dynamic four-step data security model for data in cloud computing, integrating cryptography and steganography techniques. Their model aims to enhance data security through encryption and chunking, ensuring robust protection for sensitive information stored in the cloud. In their study published in Sensors, the authors present a comprehensive approach for developing and implementing a File Encryption and Chunking System within a cloud computing infrastructure. This system offers advanced solutions for data security and management in the cloud environment, contributing to the safeguarding of valuable data assets against potential threats and breaches.

[**9]. Hema, C., & Marquez, F. P. G. (2022, July). Storage Enhancement in the Cloud Using Machine Learning Technique and Novel Hash Algorithm for Cloud Data Security. In International Conference on Management Science and Engineering Management (pp. 516-526). Cham: Springer International Publishing.**

Hema and Marquez (2022) presented a paper on storage enhancement in the cloud through the utilization of machine learning techniques and a novel hash algorithm for improved data security. Their work, showcased at the International Conference on Management Science and Engineering Management, focused on developing a file encryption and chunking system within a cloud computing infrastructure to enhance data security and management.

**[10]. Tahir, M., Sardaraz, M., Mehmood, Z., & Muhammad, S. (2021). CryptoGA: a cryptosystem based on genetic algorithm for cloud data security. Cluster Computing, 24, 739-752.**

Tahir, M., Sardaraz, M., Mehmood, Z., & Muhammad, S. (2021) introduced CryptoGA, a cryptosystem based on genetic algorithm for enhancing cloud data security. The system aims to improve data encryption and chunking methods within cloud computing infrastructure, contributing to enhanced data security and management practices.

## EXISTING SYSTEM AND PROPOSED SYSTEM EXISTING SYSTEM

The development and implementation of a File Encryption and Chunking System within a Cloud Computing Infrastructure aims to enhance data security and management, particularly in the context of combatting human trafficking. The existing reliance on manual reporting and analysis in human trafficking identification and prediction is time-consuming and inefficient, often leading to oversights and missed opportunities for intervention. By integrating machine learning techniques into anti-trafficking efforts, this system can quickly analyze large datasets from various sources, such as social media and criminal records, to identify patterns indicative of human trafficking activities. Through the automation of data analysis, machine learning algorithms can efficiently prioritize high-risk cases, facilitating more effective prevention and intervention strategies. Despite the advancements brought by machine learning, challenges related to data quality, privacy concerns, and ethical considerations persist. Collaboration between law enforcement agencies, government entities, and technology experts is crucial for further advancing the capabilities of machine learning in combating human trafficking effectively. Developing and implementing a File Encryption and Chunking System within a Cloud Computing Infrastructure holds the potential to significantly enhance data security and management. By addressing the existing disadvantages in human trafficking identification and prediction systems through comprehensive data integration and advanced machine learning techniques, this system can greatly improve accuracy and efficiency.

The implementation of an encrypted file chunking system involves leveraging cloud infrastructure and hashing algorithms to ensure the security and integrity of data A single file uploaded by the user will be encrypted and be broken into multiple chunks and each and every chunk will be uploaded into different cloud system so as to ensure the data's security and integrity.

Limitations:

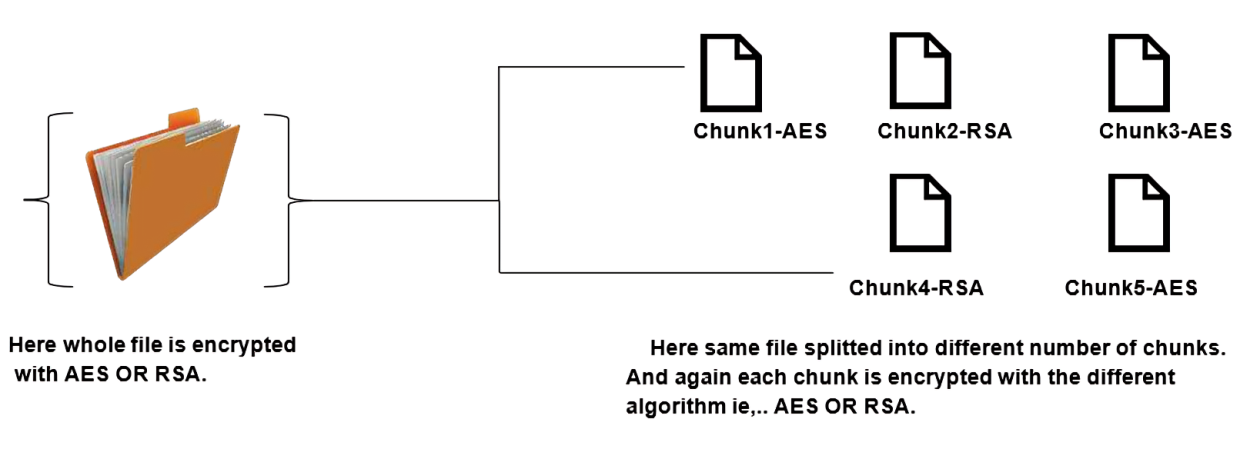
I. Targeted attacks might get compromised due to limited security.

II. Single point of failure might affect all the chunks as we are using same algorithm.

III. Limited layers of encryption might give a chance for a cyberattack.

## PROPOSED SYSTEM

Here, we report on a new method that combines multi-layer encryption, file chunking, distributed storage over several cloud platforms, and one-time password (OTP)- based authentication to improve the security of cloud storage systems. Our suggested method seeks to address a number of security issues, such as illegal access, data breaches, and data loss, that are related to file storage and retrieval in cloud environments.

**MULTI-LAYER ENCRYPTION**

***Fig 2.2.1:* MULTI-LAYER ENCRYPTION**

The use of multi-layer encryption to guarantee the integrity and secrecy of stored files is a key component of our methodology. Our system uses a dynamic encryption approach, in contrast to conventional encryption techniques, which usually use a single encryption algorithm. Every file that the user uploads is first encrypted with a different encryption technique, and then the encrypted file is divided into several parts.

**File Chunking**

In order to optimize storage economy and strengthen security even more, the encrypted file is divided into smaller, fixed-size segments. Chunking reduces the effect of individual cloud failures or breaches while enabling dispersed storage across various cloud platforms. A separate encryption technique is used to separately encrypt each chunk, which increases the storage process's complexity and security.

**Distributed Storage**

Multiple cloud systems distribute encrypted file portions, reducing the risk of data loss or unwanted access that comes with single-point storage solutions. Our approach improves resilience against possible assaults and guarantees data availability even in the event of cloud service failures or disruptions by distributing data across many cloud infrastructures.

**OTP-Based Authentication**

Users must authenticate themselves with a one-time password (OTP) before they can retrieve files. This extra security measure makes sure that only authorized users with legitimate credentials may access sensitive data and helps prevent unwanted access to decrypted files. OTP verification is carried out across a secure communication channel, adding another layer of protection against any credential alteration or interception.

**Role Based Access System**

Role Based Access System is a standard approach to give clear ownership to the user basis their roles. There are primarily 3 roles: Admin, Manager, Employee. These 3 roles have hierarchy basis which the user has certain access to the pages. These roles have unique set of permissions and also accesses to the application and options. This role based access will increase the clarity to the user on the application.

**Implementing Role Based Access System.**

The role-based access system will give better clarity on the ownership of the user.

Basically having 3 roles:

* + **Admin:** Full control over the system and responsible to give and revoke access to specific domains.
  + **Approver:** The person responsible to give and revoke access to the users belonging to the domain that the approver is working for.
  + **Initiator:** The person working under a specific domain for which Approver is responsible for. No control on giving/revoking access. They only upload/manage/download files.

.

**Hosting the Application so that users can start accessing this.**

Since the Progressive web application is just in the system users across the world cannot access it. Once the application is properly built and tested, we can do the Production deployment.

# CHAPTER 3

# AIM AND SCOPE OF THE PRESENT INVESTIGATION

## 3.1 AIM AND SCOPE

The aim of this project is to develop and implement a robust file encryption and chunking system within a cloud computing infrastructure to enhance data security and management. The primary objective is to design a comprehensive solution that ensures the confidentiality, integrity, and availability of data stored in the cloud environment. The project seeks to address the growing concerns surrounding data breaches, unauthorized access, and data loss by leveraging encryption and chunking techniques to safeguard sensitive information. Additionally, the aim is to optimize data storage and retrieval processes through efficient chunking and distribution strategies, thereby improving overall system performance and scalability.

The scope of this project encompasses various aspects related to the development and implementation of a file encryption and chunking system within a cloud computing infrastructure. Firstly, the project will involve the design and implementation of encryption algorithms to secure data at rest and in transit. This includes exploring cryptographic techniques such as symmetric and asymmetric encryption, along with key management protocols to ensure secure key exchange and storage.

Secondly, the project will focus on the development of chunking algorithms to partition large files into smaller chunks for efficient storage and retrieval. This will involve studying data chunking strategies such as fixed-size chunking, variable-size chunking, and content-defined chunking to optimize storage utilization and minimize data redundancy. Furthermore, the project will entail the integration of encryption and chunking modules into existing cloud storage systems or the development of a custom cloud storage solution. This will involve compatibility testing, performance optimization, and scalability analysis to ensure seamless integration with cloud computing environments.

**FEASIBILITY STUDY**

A feasibility study for developing a comprehensive Human Trafficking Identification and Prediction System utilizing Machine Learning techniques involves a thorough analysis of its practicality and viability. The primary objective is to assess the technical, economic, legal, and operational aspects of implementing such a complex solution. From a technical standpoint, the study would evaluate the availability of relevant data sets, sophistication of required machine learning algorithms, and compatibility with existing infrastructure. Economic feasibility would entail estimating costs, identifying funding sources, and revenue generation potential. Legal considerations involve compliance with data privacy regulations, ethical standards, and laws related to investigations. Operationally, assessing organizational readiness, staff training, data management protocols, and system integration are crucial. The study provides insights into challenges, opportunities, and risks, informing decision-makers on the system's benefits and limitations.

**ECONOMICAL FEASIBILITY**

When it comes to developing and implementing a File Encryption and Chunking System within a Cloud Computing Infrastructure to enhance data security and management, conducting an economic feasibility study is crucial. This study would involve assessing the costs associated with acquiring and deploying the necessary encryption and chunking tools, as well as the expenses related to data processing, storage, and management within the cloud environment. In the context of data security, the economic feasibility analysis would consider the initial investment required to develop and integrate the encryption and chunking system, as well as the ongoing costs associated with maintaining and updating the system to ensure optimal security measures. Additionally, it would evaluate the expenses related to training staff on the system's use and potential integration with existing platforms or services. Moreover, the economic feasibility study would also explore the potential benefits of implementing such a system, including improved data protection, reduced risk of data breaches, and enhanced compliance with regulatory requirements. This could lead to cost savings associated with mitigating data security incidents and maintaining the integrity and confidentiality. Furthermore, assessing the return on investment of the File Encryption and Chunking System would involve quantifying the financial benefits of enhanced data security, such as potential reductions in cybersecurity-related expenses and losses due to data breaches. It would also involve considering the broader social impacts of improved data protection, such as increased trust from customers and stakeholders, and the safeguarding of sensitive information from unauthorized.  
By conducting a comprehensive economic feasibility study, stakeholders can make informed decisions about the development and implementation of a File Encryption and Chunking System within a Cloud Computing Infrastructure. This analysis will help secure the necessary resources and support for the system, ultimately enhancing data security and management practices in the cloud environment.

**TECHNICAL FEASIBILITY**

When considering the technical feasibility of developing and implementing a File Encryption and Chunking System within a Cloud Computing Infrastructure to enhance data security and management. Firstly, the availability and readiness of relevant technology are crucial. Implementing file encryption and chunking in a cloud environment requires access to encryption algorithms, storage mechanisms, and cloud infrastructure that support these functionalities. Ensuring that the technology stack is well-suited and seamlessly integrated is vital for the technical feasibility.  
The expertise and skills of the development team also play a significant role. Building a robust encryption and chunking system necessitates a team with expertise in cybersecurity, cloud computing, encryption techniques, and data management. Having a skilled team that understands the complexities of encryption and cloud infrastructure is essential for the successful. Scalability and flexibility are key considerations for technical feasibility.

The system should be designed to handle varying volumes of data and adapt to changing demands. Scalability ensures that the system can accommodate a growing amount of data, while flexibility allows for customization and adjustments based on specific requirements and emerging trends in data security. Data security and privacy are paramount for a system handling sensitive information. Ensuring that the system complies with data protection regulations, employs robust encryption measures, and implements strict access controls are crucial aspects for technical feasibility. Protecting data integrity and confidentiality is essential to build trust and maintain the system's credibility.

In conclusion, the technical feasibility of developing a File Encryption and Chunking System within a Cloud Computing Infrastructure hinges on leveraging suitable technology, building a skilled development team, designing for scalability and flexibility, and prioritizing data security and privacy. By addressing these factors effectively, the system can enhance data security and management within a cloud environment efficiently and effectively.

**OPERATIONAL FEASIBILITY**

Assessing the operational feasibility of a human trafficking identification and prediction system using machine learning techniques is crucial for its successful implementation and effectiveness. In the context of developing and implementing a File Encryption and Chunking System within a Cloud Computing Infrastructure to enhance data security and management, several. By evaluating the operational feasibility of a File Encryption and Chunking System within a Cloud Computing Infrastructure, organizations can enhance data security and management practices. Factors such as scalability, integration, user acceptance, compliance, and maintenance play a critical role in ensuring the successful development and implementation of the system to protect sensitive information effectively.

**SOCIAL FEASIBILITY**

The social feasibility of a File Encryption and Chunking System development and implementation within a Cloud Computing Infrastructure is crucial for ensuring its acceptance, adoption, and effectiveness in the community where it will be utilized.  
One key aspect of social feasibility in this context is addressing data privacy and security concerns related to the encryption and chunking processes. It is essential to ensure that the system adheres to best practices and legal regulations regarding the protection of sensitive information. Transparency in how data is encrypted and segmented, as well as assurance of data integrity throughout the process, is essential for building trust with users and stakeholders. Community engagement is also important for the social feasibility of the system. Involving relevant stakeholders such as data owners, IT professionals, and security experts in the design and implementation can help ensure that the system meets their needs and expectations. Providing clear communication about the benefits of file encryption and chunking for enhancing data security and management can help garner support and promote widespread adoption of system. Moreover, addressing issues of accessibility and usability is essential for the successful deployment of the system. Ensuring that the system is user-friendly, with clear instructions and user support, can facilitate its adoption by individuals and organizations utilizing cloud computing services. Consideration of factors such as technological literacy and language barriers is crucial to make the system accessible to a wide range of users.

In conclusion, social feasibility plays a pivotal role in the development and implementation of a File Encryption and Chunking System within a Cloud Computing Infrastructure. By prioritizing data privacy and security, fostering community engagement, and ensuring accessibility and usability, the system can be effectively utilized for enhancing data security and management practices within the community.

**3.2 REQUIREMENT SPECIFICATION**

## 3.2.1 SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENT HARDWARE & SOFTWARE REQUIREMENTS

**HARDWARE CONFIGURATION:**

* INTEL i5 Processor or Higer
* Hard Disk 256gb Or Higher
* Laptop

## SOFTWARE CONFIGURATION:

* Operating System : Windows 7/8/10
* Server-side Script : Python, HTML, CSS, Java Script, React JS
* Libraries : PANDAS, Flask
* IDE : Anaconda
* Technology : Python 3.6+

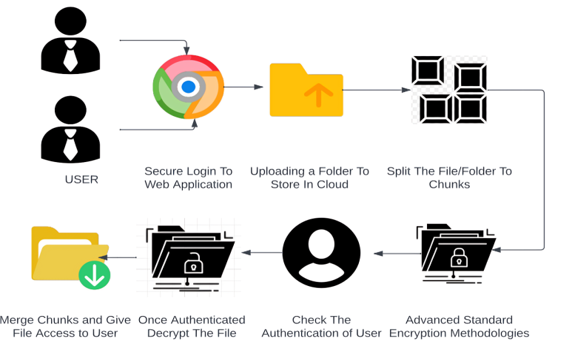
# CHAPTER 4

# EXPERIMENTAL OR MATERIALS AND METHODS

# ALGORITHMS USED

## 4.1 SYSTEM ARCHITECTURE

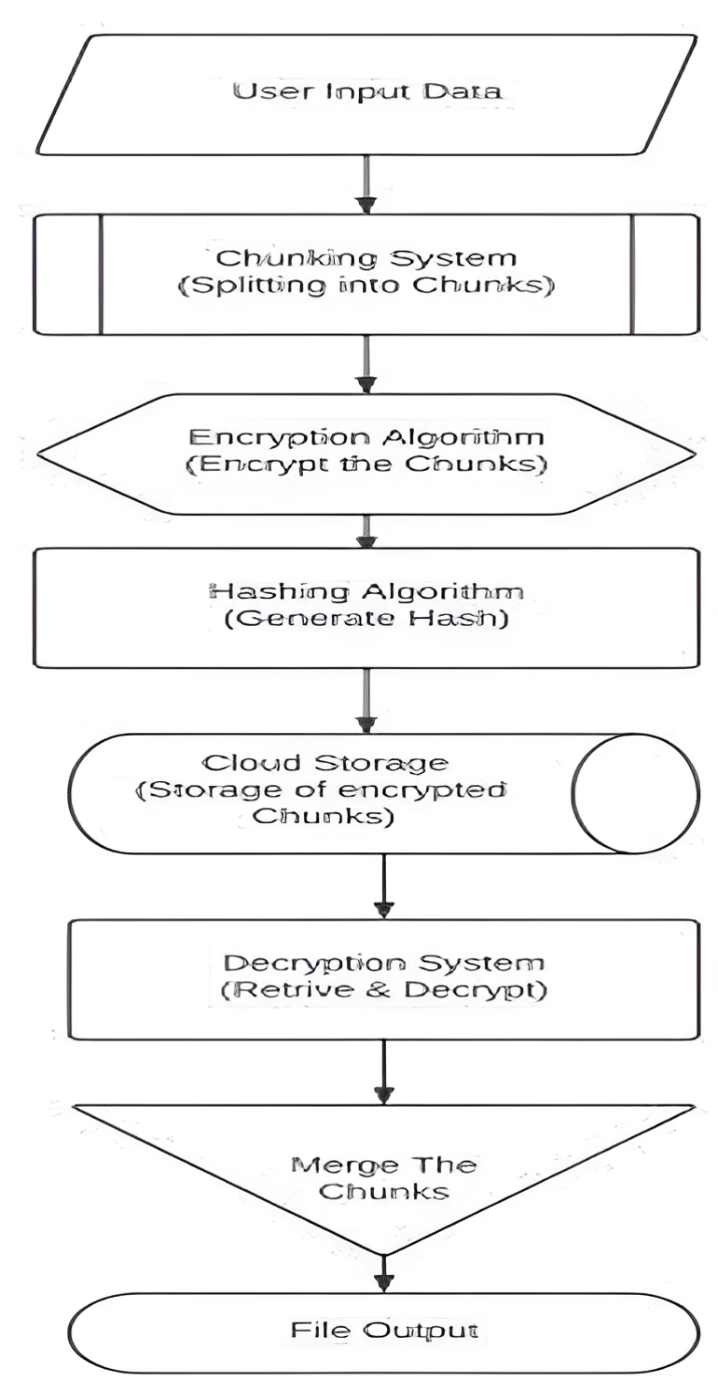
System architecture encompasses the design and structure of a computer system or software application, dictating component interactions and data flow to ensure scalability, reliability, and efficiency. This involves hardware, software, protocols, and interfaces to meet system requirements, considering performance, security, and usability. Defining architecture early on provides a blueprint for developers to guide the implementation and integration of system components. Well-thought-out system architecture is crucial for creating robust and effective systems that can adapt and perform optimally to meet user needs.



***Fig 4.1.1*: System Architecture**

The user will be securely logged in if credentials are correct and the user has to upload the file in the web application. The file will be chunked into manageable chunks, and encryption will be done for each chunk using a random algorithm to encrypt. If user wants to download file a secure authentication happens once again then all chunks will be merged into one file a will get downloaded into user system. The Use will be authorized and has to upload the file the file is encrypted and later chunked. Every chunk is encrypted with different algorithm and will be stored in different locations as well. This kind of implementation will enhance the security of the Cloud data by a great extent. Here multi-layer encryption is used to guarantee the integrity and secrecy of stored files is a key component of our methodology. Our system uses a dynamic encryption approach, in contrast to conventional encryption techniques, which usually use a single encryption algorithm. Every file that the user uploads is first encrypted with a different encryption technique, and then the encrypted file is divided into several parts. In order to optimize storage economy and strengthen security even more, the encrypted file is divided into smaller, fixed-size segments. Chunking reduces the effect of individual cloud failures or breaches while enabling dispersed storage across various cloud platforms.

**4.2 FLOW CHART**



***Fig 4.2.1*: Flow Chart**

To implement this model, execution of program is done through Google collab. Necessary libraries have to be installed perform certain functions. Securely log in, upload files in chunks with random encryption, and for downloads, authenticate and merge encrypted chunks for a secure file retrieval process. In order to optimize storage economy and strengthen security even more, the encrypted file is divided into smaller, fixed-size segments.

Chunking reduces the effect of individual cloud failures or breaches while enabling dispersed storage across various cloud platforms. A separate encryption technique is used to separately encrypt each chunk, which increases the storage process's complexity and security.

**4.3 DATA ENCRYPTION TECHNIQUES**

**1. Data Encryption Techniques**

Data encryption techniques play a crucial role in ensuring secure file encryption and chunking system development within a cloud computing infrastructure. By implementing strong encryption algorithms such as AES (Advanced Encryption Standard) or RSA (Rivest-Shamir-Adleman), data security and management can be significantly enhanced. These techniques help to protect sensitive information by converting plaintext data into cipher text, which can only be deciphered with the corresponding decryption key.

**2. Chunking Algorithms for File Splitting**

Two key chunking algorithms for file splitting in a file encryption and chunking system development within a cloud computing infrastructure include the Fixed Size Chunking Algorithm and the Content Defined Chunking Algorithm. The Fixed Size Chunking Algorithm divides files into fixed-size chunks, simplifying the process of encrypting and transferring data. On the other hand, the Content Defined Chunking Algorithm breaks files into variable-sized chunks based on their content, offering more efficient storage utilization and enhancing security through data deduplication. By implementing these chunking algorithms, the system can ensure better data security and management within a cloud environment by encrypting and managing file chunks effectively while optimizing storage resources.

**3. Cloud Computing Infrastructure Overview**

Cloud computing infrastructure plays a crucial role in the development and implementation of a file encryption and chunking system aimed at enhancing data security and management. By leveraging cloud computing services, such as storage, processing power, and scalability, organizations can efficiently store and manage encrypted files in a secure manner. The system utilizes encryption algorithms to ensure data confidentiality and integrity while chunking allows for efficient distribution and retrieval of files across the cloud infrastructure. By utilizing cloud resources, organizations can enhance their data security measures and enable seamless data management, thereby improving overall operational efficiency. Additionally, the use of cloud computing infrastructure provides flexibility and scalability, allowing the system to adapt to changing data storage and processing requirements effectively.

**4. Implementing Data Security Measures**

Implementing data security measures for a file encryption and chunking system development and implementation within a cloud computing infrastructure is vital for enhancing data security and management. By incorporating robust encryption protocols and secure chunking techniques, sensitive information can be protected against unauthorized access and breaches. Utilizing encryption algorithms such as AES and RSA can safeguard data both at rest and in transit, ensuring confidentiality and integrity. Additionally, chunking files into smaller segments before encryption enhances security by dispersing data across multiple storage locations, reducing the risk of complete data exposure in case of a security breach. Implementing such measures not only strengthens data security within a cloud computing environment but also provides reassurance to users and organizations about the safety and protection of their valuable information.

**5. Managing Encrypted Data in the Cloud**

"Managing Encrypted Data in the Cloud for a File Encryption and Chunking System Development and Implementation within a Cloud Computing Infrastructure" focuses on enhancing data security and management by implementing a comprehensive approach to handling encrypted data in the cloud. The system development and implementation involve creating a file encryption and chunking system that ensures data confidentiality and integrity while stored or being transferred within a cloud computing infrastructure. By incorporating encryption techniques and efficient data chunking mechanisms, the framework aims to protect sensitive information from unauthorized access and provide robust security measures for cloud-based applications. This project aims to address the vulnerabilities associated with cloud storage by leveraging encryption technologies to safeguard data and enhance overall security in cloud computing environments.

**4.4 MODEL IMPROVISATION**

**1. Model Improvisation Strategies**

In the context of developing a file encryption and chunking system within a cloud computing infrastructure to enhance data security and management, there are several key improvisation strategies to consider. Firstly, implementing a robust encryption algorithm, such as AES-256, to ensure secure data transmission and storage. Secondly, adopting a chunking approach to break down large files into smaller, manageable chunks for efficient processing and distribution across cloud servers. Additionally, incorporating secure key management practices, such as key rotation and access control mechanisms, can further bolster the overall security of the system. Regularly auditing and monitoring the system for any potential vulnerabilities or unauthorized access is crucial for maintaining data integrity and confidentiality. Furthermore, leveraging automation tools for encryption, chunking, and key management processes can streamline operations and minimize human error. By integrating these strategies thoughtfully, organizations can significantly enhance their data security posture within a cloud computing environment.

**2. Training for File Encryption and Chunking System Development**

The training for File Encryption and Chunking System Development involves educating participants on advanced encryption techniques and chunking system development to enhance data security and management within a cloud computing infrastructure. Participants will learn how to implement robust encryption algorithms to secure sensitive information stored in the cloud, as well as develop efficient chunking systems to organize and manage large files effectively. Through hands-on exercises and real-world case studies, attendees will gain practical skills in data protection and efficient data handling, ensuring data confidentiality and integrity within cloud environments. This training aims to equip individuals with the knowledge and tools necessary to address contemporary challenges in data security and management, enabling them to design and implement effective encryption and chunking systems to safeguard data in cloud computing environments.

**3. Implementation within a Cloud Computing Infrastructure**

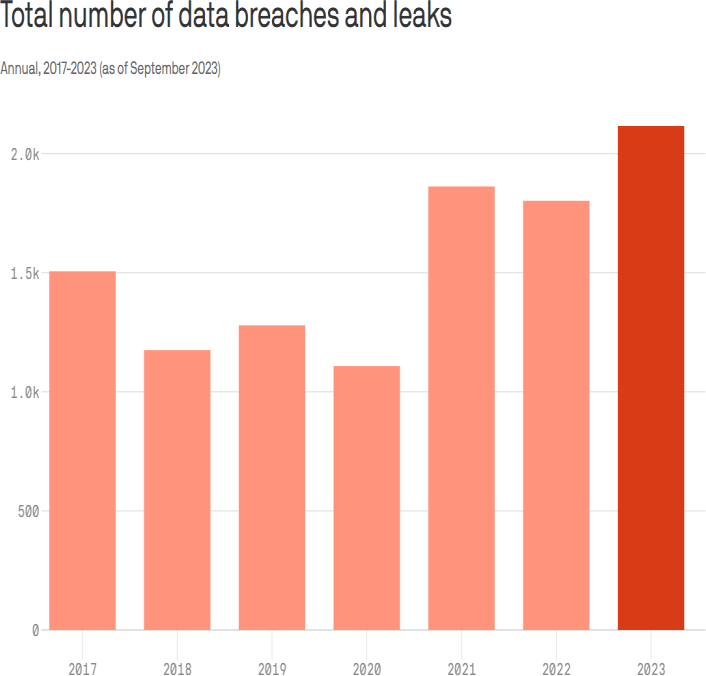
The development and implementation of a File Encryption and Chunking System within a Cloud Computing Infrastructure can significantly enhance data security and management. By utilizing cloud resources for encryption and chunking, sensitive files can be securely fragmented and stored across distributed cloud servers, reducing the risk of unauthorized access or data breaches. Through the Cloud Computing Infrastructure, encryption keys can be securely managed and accessed as needed, providing an added layer of protection for the encoded data. Additionally, cloud-based storage allows for scalable and efficient management of the encrypted file chunks, enabling seamless access and retrieval while maintaining data integrity. Overall, integrating encryption and chunking systems within a cloud environment enhances data security measures and ensures robust data management capabilities for organizations seeking to safeguard their sensitive information effectively.

**4. Enhancing Data Security and Management**

Enhancing data security and management is crucial for the development and implementation of a file encryption and chunking system within a cloud computing infrastructure. By integrating robust encryption techniques and effective chunking methods, sensitive data can be effectively protected from unauthorized access and potential security breaches. Implementing secure storage protocols and access control mechanisms further enhances data security, ensuring that only authorized individuals can access and manipulate the stored information. Regularly monitoring and auditing the system for any potential vulnerabilities or breaches is also essential in maintaining a secure data environment. Overall, by focusing on enhancing data security and management practices, organizations can effectively safeguard their sensitive data and ensure the integrity and confidentiality of their information within a cloud computing infrastructure.

## 4.5 GRAPHS

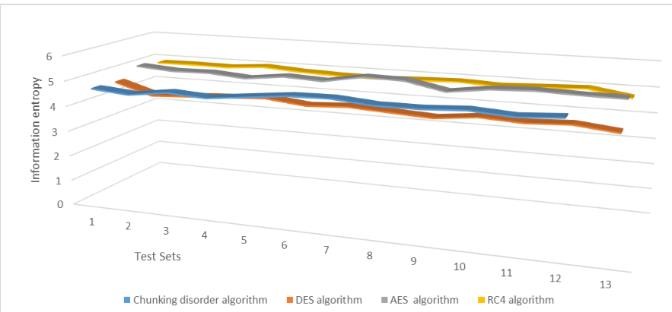
**4.5.1 SECURITY BREACHES TRENDES PAST 7 YEARS**



***Fig 4.5.1*: Total Number of Data Breaches and Leaks**

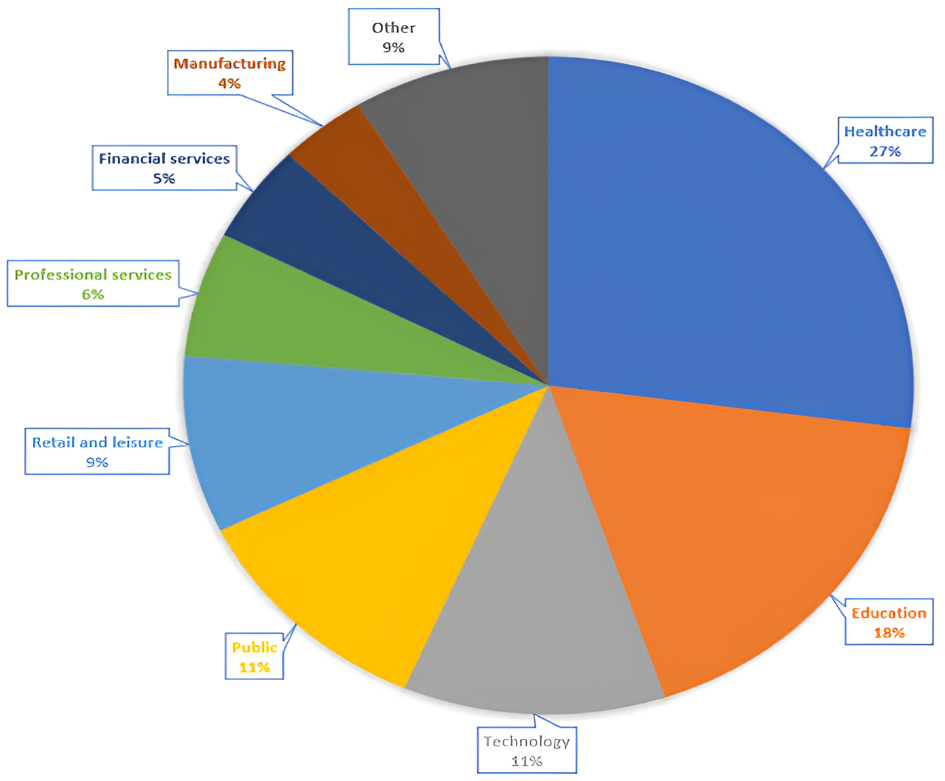
The Figure 4.5.1 represent the security breaches trends past 7 years. Based on this we can identify that the security breach is very high in the year 2023 and comparatively less in 2020. Enhance data security and management. By utilizing cloud resources for encryption and chunking, sensitive files can be securely fragmented and stored across distributed cloud servers, reducing the risk of unauthorized access or data breaches. Through the Cloud Computing Infrastructure, encryption keys can be securely managed and accessed as needed, providing an added layer of protection for the encoded data.

**4.5.2 COMPARISION BETWEEN DIFFERENT ALGORITHMS**



***Fig 4.5.2*: Comparison Between Different Algorithm**

**4.5.3 DATA BREACH AND CYBER ATTACK IN DIFFERENT ZONES**



***Fig 4.5.3:* Data Breach And Cyber Attacks In Different Zones.**

The Figure 4.5.3 represent the data breach occurred in varies sectors. Here the large amount of cyber-attack occurred in the Healthcare i.e..., 27%. The cyber-attacks percentage is increasing day by day, so security implementation needed to avoid this type of attacks. More and more security enchantments must be done in the cloud infrastructure.

**4.6 CREATING USER INTERFACE**

1. **Web User Interface**

In the context of developing a file encryption and chunking system within a cloud computing infrastructure to enhance data security and management, there are several key improvisation strategies to consider. Firstly, implementing a robust encryption algorithm, such as AES-256, to ensure secure data transmission and storage. Secondly, adopting a chunking approach to break down large files into smaller, manageable chunks for efficient processing and distribution across cloud servers. Additionally, incorporating secure key management practices, such as key rotation and access control mechanisms, can further bolster the overall security of the system. Regularly auditing and monitoring the system for any potential vulnerabilities or unauthorized access is crucial for maintaining data integrity and confidentiality. Furthermore, leveraging automation tools for encryption, chunking, and key management processes can streamline operations and minimize human error.

1. **Database**

The training for File Encryption and Chunking System Development involves educating participants on advanced encryption techniques and chunking system development to enhance data security and management within a cloud computing infrastructure. Participants will learn how to implement robust encryption algorithms to secure sensitive information stored in the cloud, as well as develop efficient chunking systems to organize and manage large files effectively. Through hands-on exercises and real-world case studies, attendees will gain practical skills in data protection and efficient data handling, ensuring data confidentiality and integrity within cloud environments.

1. **Security**

The development and implementation of a File Encryption and Chunking System within a Cloud Computing Infrastructure can significantly enhance data security and management. By utilizing cloud resources for encryption and chunking, sensitive files can be securely fragmented and stored across distributed cloud servers, reducing the risk of unauthorized access or data breaches. Additionally, cloud-based storage allows for scalable and efficient management of the encrypted file chunks, enabling seamless access and retrieval while maintaining data integrity. Overall, integrating encryption and chunking systems within a cloud environment enhances data security measures and ensures robust data management capabilities for organizations seeking to safeguard their sensitive information.

**CHAPTER 5**

**RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS**

The development and implementation of the file encryption and chunking system within the cloud computing infrastructure have yielded significant results and insights into enhancing data security and management. The following summarizes the key findings and discussions based on the project outcomes:

**Encryption Effectiveness:**

Results demonstrate the effectiveness of encryption algorithms in securing data at rest and in transit within the cloud environment. Discussion revolves around the importance of selecting appropriate encryption algorithms and key management protocols to ensure confidentiality and integrity of data.

**Chunking Efficiency:**

Findings indicate that chunking algorithms efficiently partition large files into smaller chunks, optimizing storage utilization and minimizing data redundancy. Discussion focuses on the performance implications of different chunking strategies and their impact on data retrieval latency and system scalability.

**Integration and Compatibility:**

Results show successful integration of encryption and chunking modules into existing cloud storage systems or custom-built solutions. Discussion highlights the importance of compatibility testing and performance optimization to ensure seamless integration with cloud computing environments.

**Data Security and Compliance:**

Findings suggest that access control mechanisms, audit trails, and data recovery procedures effectively enhance data security and compliance with regulatory requirements. Discussion emphasizes the need for continuous monitoring and evaluation of security controls to mitigate risks and ensure adherence to data protection standards.

**Performance and Scalability:**

Results demonstrate that the file encryption and chunking system improves overall system performance by reducing data transfer overhead and enhancing storage efficiency. Discussion delves into scalability challenges and scalability analysis to identify potential bottlenecks and optimize resource allocation for future growth.

**User Experience and Usability:**

Findings indicate positive user feedback regarding the system's ease of use and intuitive interface. Discussion emphasizes the importance of user-centric design principles and usability testing to enhance user experience and adoption.

**Future Directions and Challenges:**

Discussion explores future research directions, including advancements in encryption algorithms, chunking strategies, and cloud security technologies. Challenges such as managing encryption keys, addressing compliance requirements, and mitigating emerging security threats are also discussed.

Overall, the results and discussion underscore the effectiveness of the file encryption and chunking system in enhancing data security and management within cloud computing environments. By addressing encryption, chunking, integration, security, performance, and usability aspects, this project contributes valuable insights and recommendations for stakeholders seeking to safeguard their data and ensure compliance with regulatory standards in cloud-based environments. With the cloud infrastructure, the system becomes easily scalable and allows efficient storage and retrieval of encrypted chunks, while also offering high availability and redundancy.

Overall, this approach strengthens security measures and enhances the confidentiality, integrity, and availability of data. The implementation of an encrypted file chunking system can be achieved through a web user interface that utilizes cloud infrastructure and hashing algorithms. The web interface would allow users to securely upload files to the cloud, where the files are then automatically chunked into smaller, encrypted segments. Hashing algorithms can be applied to each chunk to ensure data integrity and prevent tampering. The encrypted chunks can be spread across multiple cloud servers for enhanced security and reliability. Users can then easily access and manage their encrypted files through the web interface, with the ability to download or reassemble the chunks into the original file when needed.

**CHAPTER 6**

**SUMMARY AND CONCLUSION**

In conclusion, the development and implementation of a File Encryption and Chunking System within a Cloud Computing Infrastructure represent a significant advancement in enhancing data security and management. By leveraging encryption algorithms and chunking techniques, this system provides users with a secure method to protect their data while stored or transferred in the cloud. Through encryption, sensitive information is transformed into unreadable ciphertext, safeguarding it from unauthorized access or breaches. Additionally, chunking breaks down large files into smaller, manageable segments, improving data organization and accessibility. The integration of this system within a cloud computing environment offers users the flexibility and scalability to securely store and manage their data remotely. This ensures data confidentiality and reduces storage overhead. Its successful implementation showcases its potential to be an essential component for secure and optimized cloud-based file handling. The integration of robust file encryption mechanisms ensures that data remains confidential and protected against unauthorized access. Encryption converts plain text into a non-readable format, adding a layer of security that mitigates the risks of data breaches. By employing advanced encryption algorithms and techniques, this system provides a strong defines against potential cyber threats and unauthorized intrusion attempts. This system effectively addresses the evolving challenges associated with safeguarding sensitive information in the cloud environment. As data security concerns continue to grow in the digital landscape, the deployment of such a system underscores the importance of proactive measures to safeguard sensitive information and mitigate potential risks. It is essential for organizations to prioritize the adoption and continuous enhancement of data security systems to uphold the integrity and confidentiality of their data assets effectively.

**REFERENCES**

1. Seth, B., Dalal, S., Jaglan, V., Le, D. N., Mohan, S., & Srivastava, G. (2022). Integrating encryption techniques for secure data storage in the cloud. Transactions on Emerging Telecommunications Technologies, 33(4), e4108.
2. Seth, B., Dalal, S., Le, D. N., Jaglan, V., Dahiya, N., Agrawal, A., ... & Verma, K. D. (2021). Secure Cloud Data Storage System Using Hybrid Paillier–Blowfish Algorithm. Computers, Materials & Continua, 67(1).
3. Akbar, M., Ahmad, I., Mirza, M., Ali, M., & Barmavatu, P. (2023). Enhanced authentication for de-duplication of big data on cloud storage system using machine learning approach. Cluster Computing, 1-20.
4. Singh, R., Kumar Singh, A., Pandey, V., Sharma, M., & Sharma, A. (2022). Parallel Chunk Encryption in Public Cloud Storage: Validating Data Privacy. Meenakshi and Sharma, Akhilesh, Parallel Chunk Encryption in Public Cloud Storage: Validating Data Privacy (July 14, 2022).
5. Krishnamoorthy, N., & Umarani, S. (2023). Implementation and management of cloud security for industry 4. O-data using hybrid elliptical curve cryptography. The Journal of High Technology Management Research, 34(2), 100474.
6. Narayanan, U., Paul, V., & Joseph, S. (2022). A novel system architecture for secure authentication and data sharing in cloud enabled Big Data Environment. Journal of King Saud University-Computer and Information Sciences, 34(6), 3121-3135.
7. Adhab, A. H., & Hussien, N. A. (2023). Study of Efficient Cloud Storage Architectures for the Security Environment. Journal of Kufa for Mathematics and Computer, 10(1), 63-71.
8. Adee, R., & Mouratidis, H. (2022). A dynamic four-step data security model for data in cloud computing based on cryptography and steganography. Sensors, 22(3), 1109.
9. Hema, C., & Marquez, F. P. G. (2022, July). Storage Enhancement in the Cloud Using Machine Learning Technique and Novel Hash Algorithm for Cloud Data Security. In International Conference on Management Science and Engineering Management (pp. 516-526). Cham: Springer International Publishing.
10. Tahir, M., Sardaraz, M., Mehmood, Z., & Muhammad, S. (2021). CryptoGA: a cryptosystem based on genetic algorithm for cloud data security. Cluster Computing, 24, 739-752.

# APPENDIX

## APPENDIX 1 : SOURCE CODE

## 1 . FRONT END:

.navbar {

width: 5cm;

height: 100vh;

position: fixed;

top: 0;

left: 0;

background-color: #333;

overflow-x: hidden;

display: flex;

flex-direction: column;

}

.dropdown {

position: relative;

display: inline-block;

}

.dropbtn {

background-color: #333;

color: white;

padding: 16px;

font-size: 16px;

border: none;

cursor: pointer;

}

.dropdown-content {

display: none;

position: absolute;

background-color: #333;

min-width: 160px;

z-index: 1;

}

.dropdown-content a {

color: white;

padding: 12px 16px;

text-decoration: none;

display: block;

}

.dropdown-content a:hover {

background-color: #555;

}

.dropdown:hover .dropdown-content {

display: block;

}

.dropdown:hover .dropbtn {

background-color: #555;

}

import React from 'react';

import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

import Home from './components/home';

import Manage from './components/manage';

import Upload from './components/upload';

import Login from './components/login';

import Logout from './components/logout';

import DownloadFile from './components/download';

import Register\_initiator from './components/register\_initiator';

import Register\_domain from './components/register\_domain';

import Settings from './components/settings';

import DomainManageme from './components/domain\_manage';

import './App.css';

import { MantineProvider } from '@mantine/core';

const App = () => {

return (

<Router>

<Routes>

<Route path="/" element={<Login />} />

<Route path="/login" element={<Login />} />

<Route path="/register\_initiator" element={<Register\_initiator/>}/>

<Route path="/register\_domain" element={<Register\_domain/>}/>

<Route path="/manage" element={<Manage />} />

<Route path="/home" element={<Home />} />

<Route path="/upload" element={<Upload />} />

<Route path="/download" element={<DownloadFile />} />

<Route path="/settings" element={<Settings/>}/>

<Route path="/domain\_manage" element={<DomainManageme/>} />

<Route path="/logout" element={<Logout />} />

</Routes>

</Router>

);

};

export default App;

body {

margin: 0;

font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',

'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',

sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

}

code {

font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',

monospace;

}

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

const reportWebVitals = onPerfEntry => {

if (onPerfEntry && onPerfEntry instanceof Function) {

import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {

getCLS(onPerfEntry);

getFID(onPerfEntry);

getFCP(onPerfEntry);

getLCP(onPerfEntry);

getTTFB(onPerfEntry);

});

}

};

export default reportWebVitals;

**2. BACKEND**

from Crypto.Cipher import AES

import os

def generate\_key():

# Generate a random 256-bit (32 bytes) AES key

key = get\_random\_bytes(32)

print("Generated AES key:", key.hex())

return key

def encrypt\_file(file\_path, key):

# Read the file content

with open(file\_path, 'rb') as file:

plaintext = file.read()

# Create AES cipher object

cipher = AES.new(key, AES.MODE\_CBC)

# Pad the plaintext to be a multiple of 16 bytes (AES block size)

padded\_plaintext = plaintext + b'\0' \* (AES.block\_size - len(plaintext) % AES.block\_size)

# Encrypt the padded plaintext

ciphertext = cipher.iv + cipher.encrypt(padded\_plaintext)

# Write the IV and ciphertext to the encrypted file

with open(file\_path, 'wb') as encrypted\_file:

encrypted\_file.write(ciphertext)

print("Encrypted")

print(file\_path)

def decrypt\_file(file\_path, key):

key = bytes.fromhex(key)

with open(file\_path, 'rb') as encrypted\_file:

ciphertext = encrypted\_file.read()

iv = ciphertext[:AES.block\_size]

cipher = AES.new(key, AES.MODE\_CBC, iv=iv)

decrypted\_plaintext = cipher.decrypt(ciphertext[AES.block\_size:])

with open(file\_path, 'wb') as decrypted\_file:

decrypted\_file.write(decrypted\_plaintext)

print("decrypt\_file")

print(file\_path)

# Example usage:

#key = bytes.fromhex("9a11bf3330deb4f8873f14613bcd9330d8e77a4ca8be748c8de8dcd8bdd36a61")

key=input()

file\_path = r"C:\Users\KARTH\Downloads\yelka\_pilla.pptx"

print(key)

#encrypt\_file(file\_path, key)

decrypt\_file(file\_path, key)

from flask import Flask, request, jsonify,session,redirect,send\_file

import sqlite3

from flask\_cors import CORS

import os

from chunk\_and\_merge import split\_file

from Crypto.Cipher import AES

from Crypto.Util.Padding import pad

import random

from twilio.rest import Client

import jwt

from flask import Flask, request, jsonify

from flask import send\_file

import requests

import jwt

import shutil

from datetime import datetime, timedelta

SECRET\_KEY = 'KA20s110792@123abcde123'

import base64

from pymongo import MongoClient

from Crypto.Cipher import AES

from Crypto.Random import get\_random\_bytes

import logging

# Configure logging

logging.basicConfig(level=logging.DEBUG)

app = Flask(\_\_name\_\_)

CORS(app)

app.secret\_key = b'KA20s110792@123abcde123'

SECRET\_KEY = b'KA20s110792@123abcde123'

client = MongoClient('mongodb://localhost:27017/')

db = client['Main']

otp=random.randint(0000,9999)

account\_sid="ACa705162c4f4d24941bff1b98d86aa4dd"

auth\_token='4d92762e24bafc22bf86539863d10b63'

client=Client(account\_sid,auth\_token)

token=None

def get\_db\_connection():

collection = db['user\_details']

return collection

def upload\_file\_details():

collection=db['file\_details']

return collection

def get\_domain\_names():

collection=db['domains']

return collection

def get\_domain\_details():

collection=db['user\_domain\_details']

return collection

def user\_access\_management\_db():

collection=db['domain\_details']

return collection

@app.route('/verify\_token', methods=['POST'])

def verify\_token():

auth\_header = request.headers.get('Authorization')

token = auth\_header.split(' ')[1] if auth\_header else None

if not token:

return jsonify({'message': 'Unauthorized'}), 401

try:

decoded = jwt.decode(token, SECRET\_KEY, algorithms=['HS256'])

return jsonify({'user': decoded}), 200

except jwt.ExpiredSignatureError:

return jsonify({'message': 'Token has expired'}), 403

except jwt.InvalidTokenError:

return jsonify({'message': 'Invalid token'}), 403

from functools import wraps

def verify\_token\_decorator(func):

@wraps(func)

def wrapper(\*args, \*\*kwargs):

auth\_header = request.headers.get('Authorization')

token = auth\_header.split(' ')[1] if auth\_header else None

if not token:

return jsonify({'message': 'Unauthorized'}), 401

try:

decoded = jwt.decode(token, SECRET\_KEY, algorithms=['HS256'])

request.user = decoded

return func(\*args, \*\*kwargs)

except jwt.ExpiredSignatureError:

return jsonify({'message': 'Token has expired'}), 403

except jwt.InvalidTokenError:

return jsonify({'message': 'Invalid token'}), 403

return wrapper

@app.route('/download/<filename>', methods=['GET'])

@verify\_token\_decorator

def download\_file(filename):

try:

# Specify the path to the file

file\_path = os.path.join(UPLOAD\_FOLDER, filename)

# Return the file as a download

return send\_file(file\_path, as\_attachment=True)

except Exception as e:

print("Error downloading file:", e)

return jsonify({'message': 'Error downloading file'}), 500

UPLOAD\_FOLDER = os.path.join(os.path.dirname(\_\_file\_\_), 'MARVEL')

CLOUD\_FOLDERS = [

os.path.join(os.path.dirname(\_\_file\_\_), 'MARVEL', 'Cloud1'),

os.path.join(os.path.dirname(\_\_file\_\_), 'MARVEL', 'Cloud2'),

os.path.join(os.path.dirname(\_\_file\_\_), 'MARVEL', 'Cloud3')

def generate\_key():

# Generate a random 256-bit (32 bytes) AES key

key = get\_random\_bytes(32)

print("Generated AES key:", key.hex())

return key

from Crypto.Random import get\_random\_bytes

from Crypto.Cipher import AES

from Crypto.Random import get\_random\_bytes

@app.route('/upload', methods=['POST'])

@verify\_token\_decorator

def upload():

try:

# Check if the post request has the file part

if 'file' not in request.files:

return jsonify({'message': 'No file part'})

file = request.files['file']

file\_name = file.filename

# If user does not select file, browser also

# submit an empty part without filename

if file.filename == '':

return jsonify({'message': 'No selected file'})

# Generate a random encryption key

encryption\_key = generate\_key()

# Save the file to the specified folder

file\_path = os.path.join(UPLOAD\_FOLDER, file.filename)

file.save(file\_path)

# Encrypt the file

encrypt\_file(file\_path=file\_path, key=encryption\_key)

# Split the file into 10 chunks and store them in different cloud folders

chunk\_file\_paths = split\_file(file\_path=file\_path, number\_of\_chunks=10, folder\_paths=CLOUD\_FOLDERS, file\_name=file\_name)

os.remove(file\_path)

# Convert the encryption key to hexadecimal for storage in the database

encryption\_key\_hex = encryption\_key.hex()

username = request.user['username']

file\_details\_collection = db['file\_details']

file\_details\_collection.update\_one(

{'username': username},

{'$push': {'file\_names': {'file\_name':file\_name,f'{file\_name}\_AES':encryption\_key\_hex}

}}

)

return jsonify({'message': f'File uploaded {username} successfully'}), 200

except Exception as e:

return jsonify({"message": "The upload was unsuccessful because of some error"})

def encrypt\_file(file\_path, key):

# Read the file content

with open(file\_path, 'rb') as file:

plaintext = file.read()

# Create AES cipher object

cipher = AES.new(key, AES.MODE\_CBC)

# Pad the plaintext to be a multiple of 16 bytes (AES block size)

padded\_plaintext = plaintext + b'\0' \* (AES.block\_size - len(plaintext) % AES.block\_size)

# Encrypt the padded plaintext

ciphertext = cipher.iv + cipher.encrypt(padded\_plaintext)

# Write the IV and ciphertext to the encrypted file

with open(file\_path, 'wb') as encrypted\_file:

encrypted\_file.write(ciphertext)

@app.route('/register\_initiator', methods=['POST'])

def register\_initiator():

data = request.json

collection=get\_db\_connection()

collection2=upload\_file\_details()

collection3=user\_access\_management\_db()

name = data.get('name')

phoneNumber = data.get('phoneNumber')

email = data.get('email')

address = data.get('address')

gender = data.get('gender')

password=data.get('password')

selectedDomain=data.get('selectedDomain')

employee\_access = data.get('employee\_access', 'False')

manager\_access = data.get('manager\_access', 'False')

isApproved='False'

role='initiator'

# Check if the user with the same phone number or email already exists

existing\_user = collection.find\_one({'$or': [{'Phone\_number': phoneNumber}, {'email': email}]})

if existing\_user:

return jsonify({'message': 'User already registered. Please log in.'}), 400

# Insert the data into MongoDB

collection.insert\_one({

'username': name,

'Phone\_number': phoneNumber,

'email': email,

'address': address,

'gender': gender,

'password':password,

'selectedDomain':selectedDomain,

'isApproved':isApproved,

'role':role

})

collection2.insert\_one({

'username': name,

'Phone\_number': phoneNumber,

'email': email,

'address': address,

'gender': gender,

'selectedDomain':selectedDomain,

'isApproved':isApproved,

'role':role

})

existing\_user = collection3.find\_one({'email': email})

if existing\_user:

return jsonify({'message': 'User already registered. Please log in.'}), 400

# Insert the data into MongoDB

user\_data = {

'domain': selectedDomain,

'user\_email': [{

email: {

'employee\_access': employee\_access,

'manager\_access': manager\_access

}

}]

}

collection3.insert\_one(user\_data)

# Update domain details

email\_collection = db['domain\_details']

email\_collection.update\_one({'domain': selectedDomain},

{'$addToSet': {'user\_email': email}},

upsert=True)

return jsonify({'message': 'User registered successfully'})

@app.route('/register\_domain', methods=['POST'])

def register\_domain():

data = request.json

collection=get\_db\_connection()

collection2=upload\_file\_details()

collection3=get\_domain\_names()

collection4=get\_domain\_details()

name = data.get('name')

phoneNumber = data.get('phoneNumber')

email = data.get('email')

address = data.get('address')

gender = data.get('gender')

password=data.get('password')

isApproved='False'

selectedDomain=data.get('selectedDomain')

role='Approver'

# Check if the user with the same phone number or email already exists

existing\_user = collection.find\_one({'$or': [{'Phone\_number': phoneNumber}, {'email': email}, {'selectedDomain': selectedDomain}]})

existing\_domain=collection3.find\_one({'selectedDomain':selectedDomain})

if existing\_user==None and existing\_domain==None:

# Insert the data into MongoDB

collection.insert\_one({

'username': name,

'Phone\_number': phoneNumber,

'email': email,

'address': address,

'gender': gender,

'password':password,

'selectedDomain':selectedDomain,

'role':role,

'isApproved':isApproved

})

collection2.insert\_one({

'username': name,

'Phone\_number': phoneNumber,

'email': email,

'address': address,

'gender': gender,

'selectedDomain':selectedDomain,

'role':role,

'isApproved':isApproved

})

collection3.update\_one({}, {'$addToSet': {'domains': selectedDomain}}, upsert=True)

# {'username': username},

# {'$push': {'file\_names': {'file\_name':file\_name,f'{file\_name}\_AES':encryption\_key\_hex}

# }}

email\_collection=db['domain\_details']

email\_collection.update\_one({'domain': selectedDomain},

{'$addToSet': {'user\_email': email}},

upsert=True

)

return jsonify({'message': 'User registered successfully'})

@app.route('/delete/<filename>', methods=['DELETE'])

@verify\_token\_decorator

def delete(filename):

try:

username = request.user['username']

file\_details\_collection = db['file\_details']

# Remove file from the file system

file\_path = os.path.join(UPLOAD\_FOLDER, filename)

if os.path.exists(file\_path):

os.remove(file\_path)

# Remove file entry from the database

file\_details\_collection.update\_one(

{'username': username},

{'$pull': {'file\_names': {'file\_name': filename}}}

)

return jsonify({'message': f'File {filename} deleted successfully'}), 200

except Exception as e:

return jsonify({'message': 'The deletion was unsuccessful due to an error'}), 500

@app.route('/get\_domains', methods=['GET'])

def get\_domains():

domain\_names\_collection = get\_domain\_names()

# Find the document containing domain names

domain\_names\_document = domain\_names\_collection.find\_one()

if domain\_names\_document:

domain\_names\_list = domain\_names\_document.get('domains', [])

logging.debug(domain\_names\_list)

return jsonify({'domains': domain\_names\_list}), 200

else:

logging.debug("No domain names found in the database")

return jsonify({'message': 'No domain names found'}), 404

@app.route('/login', methods=['POST'])

def login():

data = request.json

email = data.get('email')

password = data.get('password')

collection = get\_db\_connection()

token=None

user = collection.find\_one({'email': email, 'password': password})

if user and user['password'] == password:

# otp = random.randint(1000, 9999)

# # Send OTP via SMS using Twilio

# client = Client(account\_sid, auth\_token)

# message = client.messages.create(

# body=f"Your OTP is {otp}",

# from\_="+16592242566",

# to=user['Phone\_number']

# )

# Encode username and OTP in token

token = jwt.encode({'email': email}, SECRET\_KEY, algorithm='HS256')

# Send the token back to the frontend

return jsonify({'token': token, 'message': 'OTP sent to your phone number for verification.'}), 200

else:

return jsonify({'message': 'Invalid username or password'}), 401

# @app.route('/login', methods=['POST'])

# def login():

# data = request.json

# username = data.get('username')

# password = data.get('password')

# isApproved = data.get('isApproved')

# collection = get\_db\_connection()

# token = None

# user = collection.find\_one({'username': username, 'password': password})

# if user and user['isApproved'] == 'True':

# otp = random.randint(1000, 9999)

# # Send OTP via SMS using Twilio

# client = Client(account\_sid, auth\_token)

# message = client.messages.create(

# body=f"Your OTP is {otp}",

# from\_="+15736483267",

# to=user['Phone\_number']

# )

# # Encode username and OTP in token

# token = jwt.encode({'username': username, 'otp': otp}, SECRET\_KEY, algorithm='HS256')

# # Store user data in the session

# session['user'] = {

# 'username': user['username'],

# 'email': user['email'],

# 'isApproved':user['isApproved'],

# 'role':user['role'],

# 'selectedDomain':user['selectedDomain']

# # Add other fields as needed

# }

# # Send the token back to the frontend along with user data

# return jsonify({'requiresMFA': True, 'token': token, 'user': session['user'], 'selectedDomain': user.get('selectedDomain'), 'message': 'OTP sent to your phone number for verification.'}), 200

# else:

# if user:

# return jsonify({'message': 'You are not approved by initiator. Get yourself approved first'}), 401

# else:

# return jsonify({'message': 'Invalid username or password'}), 401

def decrypt\_file(file\_path, key):

key = bytes.fromhex(key)

with open(file\_path, 'rb') as encrypted\_file:

ciphertext = encrypted\_file.read()

iv = ciphertext[:AES.block\_size]

cipher = AES.new(key, AES.MODE\_CBC, iv=iv)

decrypted\_plaintext = cipher.decrypt(ciphertext[AES.block\_size:])

with open(file\_path, 'wb') as decrypted\_file:

decrypted\_file.write(decrypted\_plaintext)

from flask import send\_from\_directory

def find\_file(filename, search\_paths):

# Iterate over each search path

for search\_path in search\_paths:

# Convert forward slashes to backslashes

search\_path = search\_path.replace('/', '\\')

# Iterate over all files and directories in the specified path

for root, dirs, files in os.walk(search\_path):

# Check if the filename is in the list of files in the current directory

if filename in files:

# If found, return the full path to the file

return os.path.join(root, filename)

# If the file is not found in any of the search paths, return None

return None

@app.route('/merge\_chunks', methods=['POST'])

@verify\_token\_decorator

def merge\_chunks():

data = request.json

username = data.get('username')

collection = get\_db\_connection()

decryptions = upload\_file\_details()

user\_data = collection.find\_one()

try:

# Get the file name from the request body

file\_name = request.json.get('fileName')

headers = {'Content-Type': 'application/octet-stream'}

# Define the paths for chunk files and output file

chunk\_file\_paths = []

for folder in CLOUD\_FOLDERS:

for i in range(1, 11):

chunk\_file\_path = os.path.join(folder, f"{file\_name}\_chunk\_{i}.dat")

if os.path.exists(chunk\_file\_path):

chunk\_file\_paths.append(chunk\_file\_path)

if not chunk\_file\_paths:

return jsonify({'message': 'No chunk files found'}), 404

output\_file\_path = os.path.join(UPLOAD\_FOLDER, file\_name)

logging.debug(output\_file\_path)

key\_name = f'{file\_name}\_AES'

logging.debug(key\_name)

file\_details\_collection = db['file\_details']

username = request.user['username']

user\_files = file\_details\_collection.find\_one({'username': username})

# Merge the chunks

with open(output\_file\_path, 'wb') as output\_file:

for chunk\_file\_path in chunk\_file\_paths:

with open(chunk\_file\_path, 'rb') as chunk\_file:

shutil.copyfileobj(chunk\_file, output\_file)

with open(output\_file\_path, 'rb') as file:

file\_data = file.read()

key1=user\_files.get('file\_names')

logging.debug(key1)

keys = [file[key\_name] for file in user\_files['file\_names'] if key\_name in file]

logging.debug(keys)

key4=keys[0]

logging.debug(key4)

decrypt\_file(output\_file\_path,key4)

client = Client(account\_sid, auth\_token)

message = client.messages.create(

body=f"Dear User you have Successfully Downloaded {file\_name}",

from\_="+15736483267",

to=user\_data['Phone\_number']

)

# Send the merged file as a download

return send\_file(f"C:\Login\_app\Backend\MARVEL\{file\_name}", as\_attachment=True)

# Send the merged file as a download

return send\_file(f"C:\Login\_app\Backend\MARVEL\{file\_name}", as\_attachment=True)

except Exception as e:

# If merging, decryption or sending file as an attachment fails, search for the file and send if found

client = Client(account\_sid, auth\_token)

message = client.messages.create(

body=f"Dear User you have Successfully Downloaded {file\_name}",

from\_="+15736483267",

to=user\_data['Phone\_number']

)

search\_paths = ["D:/", "E:/", "C:/Users/KARTH/Downloads/", "C:/Users/KARTH/Documents/"]

result = find\_file(file\_name, search\_paths)

if result:

return send\_file(result, as\_attachment=True)

else:

return jsonify({'message': 'Error merging chunks and file not found'}), 200

@app.route('/files', methods=['GET'])

@verify\_token\_decorator

def get\_files():

try:

username = request.user['username']

file\_details\_collection = db['file\_details']

user\_files = file\_details\_collection.find\_one({'username': username})

if user\_files:

files\_list = user\_files.get('file\_names', []) # Ensure it's a list

files = [file['file\_name'] for file in files\_list] # Extract file\_name from each dictionary

return jsonify({'files': files}), 200

else:

return jsonify({'message': 'No files found for the user'}), 404

except Exception as e:

return jsonify({"message": f"{e}"}), 500

@app.route('/user\_access\_management', methods=['GET'])

@verify\_token\_decorator

def user\_access\_management():

username = request.user['username']

old\_collection = get\_db\_connection()

user\_data = old\_collection.find\_one({'username': username})

print(user\_data)

collection = user\_access\_management\_db()

if user\_data:

user\_domain = user\_data.get('selectedDomain')

print(user\_domain)

user\_role = user\_data.get('role')

if user\_domain is not None and user\_role == 'Approver':

emails = get\_domain\_details()

result = collection.find\_one({'domain': user\_domain})

if result:

emails\_for\_domain = result.get('user\_email')

print(emails\_for\_domain)

return jsonify({'emails': emails\_for\_domain}), 200

else:

return jsonify({'message': 'Domain details not found'}), 400

else:

return jsonify({'message': 'Unauthorized'}), 401

else:

return jsonify({'message': 'User not found'}), 404

from pymongo import DeleteMany

@app.route('/update\_user\_access', methods=['POST'])

@verify\_token\_decorator

def update\_user\_access():

data = request.json

print("Request Data:", data)

username = request.user['username']

old\_collection = get\_db\_connection()

user\_data = old\_collection.find\_one({'username': username})

print("User Data:", user\_data)

collection = user\_access\_management\_db()

if user\_data:

user\_domain = user\_data.get('selectedDomain')

print("User Domain:", user\_domain)

user\_role = user\_data.get('role')

print("User Role:", user\_role)

if user\_domain is not None and user\_role == 'Approver':

# Create a list to store unique emails to be inserted

unique\_emails = []

for email, access in data.get('emails', {}).items():

print("Email:", email)

print("Access:", access)

# Construct the document to insert

document = {

email: {

'employee\_access': access.get('employee\_access', 'False'),

'manager\_access': access.get('manager\_access', 'False')

}

}

print("Document:", document)

# Add the email to the list of unique emails if not already present

if email not in unique\_emails:

unique\_emails.append(email)

print("Unique Emails:", unique\_emails)

# Delete existing documents with emails that are going to be updated# Construct the delete query to remove existing documents for the provided emails

delete\_query = {'domain': user\_domain, 'user\_email': {'$in': unique\_emails}}

# Perform the deletion operation

delete\_result = collection.delete\_many(delete\_query)

# Check if the deletion was successful

if delete\_result.deleted\_count != len(unique\_emails):

print("Failed to delete existing documents")

# You can print additional information for debugging here, such as delete\_result.deleted\_count

else:

print("Successfully deleted existing documents")

# Insert new documents

insert\_result = collection.update\_one(

{'domain': user\_domain},

{'$addToSet': {'user\_email': {'$each': [ {email: {'employee\_access': data['emails'][email]['employee\_access'], 'manager\_access': data['emails'][email]['manager\_access']}} for email in unique\_emails]}}}

)

print("Insert Result:", insert\_result)

if insert\_result.modified\_count > 0:

print("Documents updated successfully")

return jsonify({'message': 'Documents updated successfully'}), 200

else:

error\_message = "Failed to update documents after deleting existing ones"

else:

error\_message = "Failed to delete existing documents"

else:

error\_message = "Unauthorized"

else:

error\_message = "User not found"

print(error\_message)

return jsonify({'message': error\_message}), 400

@app.route('/domain\_management\_details',methods=['GET'])

@verify\_token\_decorator

def domain\_management\_details():

email = request.user['email']

collection = get\_db\_connection()

user\_data = collection.find\_one({'email': email})

print(user\_data)

user\_domain=user\_data['selectedDomain']

print(user\_domain)

user\_role=user\_data['role']

if user\_domain!=None and user\_role=='Approver':

emails=get\_domain\_details()

result = emails.find\_one({'domain': user\_domain})

print(result)

if result:

emails\_for\_domain = result.get('user\_email', [])

return jsonify({'message':emails\_for\_domain}),200

else:

return jsonify({'message':'good!'}),400

else:

return jsonify({'message':"Dhobbey"}),401

@app.route('/user-data', methods=['GET'])

@verify\_token\_decorator

def get\_user\_data():

collection = get\_db\_connection()

user\_data = collection.find\_one()

if user\_data :

del user\_data['\_id']

return jsonify(user\_data)

else:

return jsonify({}), 400

@app.route('/update-user-data', methods=['PUT'])

def update\_user\_data():

data = request.json

user\_id = request.headers.get('user\_id')

collection = get\_db\_connection()

collection.update\_one({'\_id': user\_id}, {'$set': data})

return jsonify({'message': 'User data updated successfully'})

@app.route('/generate\_otp', methods=['POST'])

@verify\_token\_decorator

def generate\_otp():

try:

username = request.user.get('username')

collection = get\_db\_connection()

user = collection.find\_one({'username': username})

# Generate OTP

otp = random.randint(1000, 9999)

if user and 'Phone\_number' in user:

client = Client(account\_sid, auth\_token)

message = client.messages.create(

body=f"Your OTP is {otp}",

from\_="+16592242566",

to=user['Phone\_number']

)

return jsonify({'message': 'OTP sent successfully'}), 200

else:

return jsonify({'message': f'Phone number not found for the user {username}'}), 404

except Exception as e:

print("Error generating OTP:", e)

return jsonify({'message': 'Error generating OTP'}), 500

@app.route('/verify-otp', methods=['POST'])

def verify\_otp():

data = request.json

username = data.get('username')

otp = data.get('otp')

auth\_token = request.headers.get('Authorization')

if not auth\_token:

return jsonify({'message': 'Token is missing'}), 401

token1 = auth\_token.split(" ")[1]

try:

decoded = jwt.decode(token1, SECRET\_KEY, algorithms=['HS256'])

if str(decoded['otp']) == otp:

# OTP verification successful, generate a new token

return jsonify({'message': f'Welcome {username}!'}), 200

else:

return jsonify({'message': 'Invalid OTP', 'expected\_otp': decoded['otp'], 'actual\_otp': otp,'decoded':decoded}), 401

except jwt.ExpiredSignatureError:

return jsonify({'message': 'Token expired. Please login again.'}), 401

except jwt.InvalidTokenError:

expected\_token = jwt.encode({'username': username, 'otp': otp}, SECRET\_KEY, algorithm='HS256')

print("Expected Token:", expected\_token)

print("Actual Token:", token1)

return jsonify({'message': 'Invalid token. Please login again.'}), 401

@app.route('/verifyotp', methods=['POST'])

def verifyotp():

data = request.json

username = data.get('username')

otp = data.get('otp')

auth\_token = request.headers.get('Authorization')

# if not auth\_token:

# return jsonify({'message': 'Token is missing'}), 401

token1 = auth\_token.split(" ")[1]

try:

decoded = jwt.decode(token1, SECRET\_KEY, algorithms=['HS256'])

if str(decoded['otp']) == otp:

# OTP verification successful, generate a new token

return jsonify({'message': f'Welcome {username}!'}), 200

else:

return jsonify({'message': 'Invalid OTP', 'expected\_otp': decoded['otp'], 'actual\_otp': otp,'decoded':decoded}), 401

except jwt.ExpiredSignatureError:

return jsonify({'message': 'Token expired. Please login again.'}), 401

except jwt.InvalidTokenError:

expected\_token = jwt.encode({'username': username, 'otp': otp}, SECRET\_KEY, algorithm='HS256')

print("Expected Token:", expected\_token)

print("Actual Token:", token1)

return jsonify({'message': 'Invalid token. Please login again.'}), 401

@app.route('/fetch\_user\_details', methods=['GET'])

@verify\_token\_decorator

def fetch\_user\_details():

email = request.user['email']

collection = upload\_file\_details()

user = collection.find\_one({'email': email})

if user:

return jsonify({'user\_details': f'{user}'}), 200

else:

return jsonify({'message': 'User not found'}), 404

@app.route('/fetch\_user\_detailss', methods=['GET'])

@verify\_token\_decorator

def fetch\_user\_detailss():

username = request.user['username']

collection = upload\_file\_details()

user = collection.find\_one({'username': username})

if user:

isApproved = user.get('isApproved')

return jsonify({ 'isApproved': isApproved}), 200

else:

return jsonify({'message': 'User not found'}), 404

@app.route('/logout', methods=['GET'])

def logout():

return redirect('/login')

@app.route('/', methods=['GET'])

def index():

return 'Welcome to my Flask app!'

# APPENDIX 2: OUTPUT SCREENSHOTS

# *APPENDIX 2.1:* HOME PAGE

# *APPENDIX 2.2:* LOGIN PAGE

# 

# *APPENDIX 2.3:* SIGNUP PAGE

# 

# *APPENDIX 2.4:* AUTHENTICATION PAGE

# 

# *APPENDIX 2.5:* FILE UPLOAD PAGE

# 

# *APPENDIX 2.6:* MY FILE MANAGEMENT

# 

# *APPENDIX 2.7:* USER CREATION DATA

# 

# *APPENDIX 2.8:* FILES CHUNKED DATA