IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

A Secure Approach To File Storage Using Fragmentation And Distributed Cloud Servers

[1]K Mohamed Faizaan, [2]D Sukumaran, [3]H Mohamed Aslam, [4] T Munnaf, [5].U Subathra

[1]Student, [2]Student, [3]Student , [4]Student, [5]Asst Professor

[1]CSE,

[1]Aalim Muhammed Salegh College of Engineering, Chennai, India

Abstract: In recent years, cyberattacks targeting sensitive systems and data have become a growing concern, despite the use of advanced encryption methods. This paper introduces a secure File Distributary System that strengthens data protection by fragmenting files into multiple encrypted segments and distributing them across various cloud platforms like AWS and Google Cloud. The program automatically selects the cloud services and securely stores each fragment without revealing the storage locations to the user. A hidden internal key is used to track and manage the fragmented data. When access is required, the system uses a secure login to retrieve and reassemble the segments into the original file. This approach enhances cybersecurity by reducing the risk of data breaches, as a hacker accessing one cloud service would only obtain an incomplete portion of the file. The paper contributes to the fields of cybersecurity, cloud storage, and distributed data protection.

Index Terms - Cloud storage security, Data confidentiality, Erasure coding, Re-encryption algorithm, Decentralized storage, Mobile application, Secure data forwarding, Encrypted communications, Distributed storage systems.

I. INTRODUCTION

Data breaches and cyberattacks have become more frequent and damaging in recent years. Even when organizations use strong encryption (a method of converting data into a secret code), attackers can sometimes find ways to break or bypass it. Once they gain access to a full, encrypted file, they may be able to decrypt it and steal sensitive information. Traditional defenses rely on storing the entire encrypted file in one place. If that single location is compromised, the attacker gets everything at once. To reduce this risk, our File Distributary System splits each file into several smaller pieces (called fragments), encrypts each fragment, and then stores them on different cloud platforms such as AWS and Google Cloud. Because each fragment by itself reveals nothing useful, an attacker who breaches one storage location cannot reconstruct the original file. A secure, hidden key (a digital "map") on a central server tracks where each fragment is stored. When an authorized user logs in—protected by a two-step verification process—the system decrypts this key, retrieves all fragments, and reassembles the file. This approach combines fragmentation (breaking data into parts) and distributed storage (using multiple cloud services) to enhance security and data availability.

II. RELATED WORKS

The primary foundation for this work is the paper by Author A and Author B, "A Novel Approach to Data Security in Cloud Storage using Erasure Coding and Re-Encryption" [1]. In that study, the authors combine erasure coding with a re-encryption mechanism to protect data stored in the cloud. They demonstrate that fragmenting data with erasure codes improves fault tolerance and that re-encryption upon each access enhances confidentiality. However, their approach requires manual

configuration of target storage nodes and exposes fragment locations to system administrators, creating potential attack vectors. Our File Distributary System builds on and extends this base by: 1. Automatic Cloud Selection – Instead of manually specifying storage targets, our system dynamically chooses among multiple cloud platforms (e.g., AWS, Google Cloud) based on availability and load. 2. Hidden Fragment Mapping – Unlike [1], which stores fragment locations in plaintext, we maintain an encrypted, centralized mapping key that is never exposed to users or operators.3. Seamless Retrieval via Secure Login – We integrate a user-transparent, two-factor authentication process to trigger decryption of the mapping key and automated reassembly of the original file, improving usability and reducing human error. ChatGPT provide conversational learning experiences. However, most of these tools are either text based or lack a structured approach for oral communication training, which is essential for improving spoken English. Recent research in human-computer interaction highlights the potential of using virtual avatars for educational purposes. Studies show that learners are more engaged when

III. SYSTEM ARCHITECTURE

The File Distributary System is composed of six core modules that work together to fragment, encrypt, distribute, and retrieve file data securely.

A. Web Interface – A Flask-based front end handles file uploads and user authentication (two-factor login).

B. Fragmentation Engine – Splits the input file into fixed-size segments.

C. Encryption Module – Applies AES-256 encryption to each fragment using unique, randomly generated keys.

D. Storage Manager – Dynamically selects target cloud platforms (AWS, Google Cloud, etc.) via each provider's API and uploads encrypted fragments

E. Key Manager – Maintains a single, encrypted internal mapping key on a centralized server; this key records each fragment's cloud location and its encryption parameters

F. Retrieval Engine – After successful authentication, decrypts the mapping key, downloads all fragments, decrypts them, and reassembles the original files

IV. PROPOSED SYSTEM

The proposed system aims to revolutionize data protection by securely splitting, encrypting, and distributing files across multiple cloud platforms. Unlike traditional storage solutions that rely on single-point encryption, this system introduces multi-layered security through intelligent file fragmentation, independent encryption, and distributed cloud storage—ensuring that no complete data resides in one place.

Key features include: The system features intelligent fragment-based file handling, where each uploaded file is automatically divided into smaller segments that are unreadable on their own. Every segment is individually secured using AES encryption, ensuring that exposure of a single fragment does not compromise the entire file. These encrypted fragments are then distributed across multiple cloud platforms such as Google Drive, AWS, and MEGA, preventing centralized data risks. A hidden, encrypted mapping key securely maintains the location and encryption details of each fragment within a centralized server. To control access, the system includes two-step authentication, incorporating OTP verification, location-based checks, or logical challenges for enhanced user verification. The entire workflow is managed through a lightweight web interface built with Flask, and future versions aim to support mobile access through an Android APK..

V. IMPLEMENTATION

A. Encryption Module

The Encryption Module uses Python's cryptography library to secure each file fragment. It supports both AES-256 (in CBC or GCM mode) and ChaCha20-Poly1305 algorithms. For every fragment, it generates a unique content-encryption key (CEK) and initialization vector, encrypts the data, and later decrypts it on retrieval. Depending on file size or user choice, the system can automatically rotate between AES and ChaCha20 to balance performance and security.

B. Key Management Service (KMS)

The Key Management Service **protects** all CEKs and fragment metadata with a single master key. This master key is itself encrypted and stored in a secure vault on the centralized server. The service maintains an encrypted JSON map that links each fragment ID to its cloud provider, object name, and encrypted CEK. Before saving to storage, the entire map is encrypted using the master key, ensuring that no sensitive mapping information is ever exposed.

C. Fragmentation Engine

The Fragmentation Engine determines how to split an uploaded file into segments. It analyzes the file's size and type, calculating optimal boundaries so that each fragment is a manageable size, while still allowing the user to override settings if desired. The engine reads the file in binary mode, divides it into slices, and sends each slice to the Encryption Module for secure processing.

D. Cloud Storage Adapter

To support multiple storage options, the system defines an abstract CloudProvider interface with upload and download methods. Concrete implementations—such as AwsS3Provider, GoogleCloudProvider, and MegaProvider—handle the specifics of each platform's SDK or API. A selection logic (e.g., round-robin, health checks, or latency measurements) decides which provider to use for each fragment, ensuring balanced load and high availability.

E. Authentication Service

Security begins with user authentication. This service offers configurable two-step verification, including one-time passwords via email or SMS, location-based checks against known IP or GPS data, and logic-based challenges like device fingerprinting. After validating credentials and the second factor, the service issues a short-lived JWT token that the user presents for all subsequent API requests.

F. Retrieval Engine

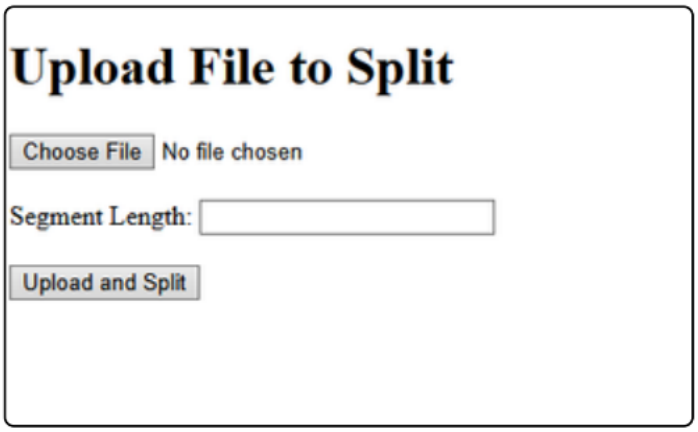
When a user requests their file, the Retrieval Engine first validates the JWT token. It then decrypts the fragment map with the master key, iterates through each entry to download and decrypt the corresponding CEK and fragment, and concatenates them in order. Finally, the original file is streamed back to the user over HTTPS, completing a secure end-to-end retrieval process.

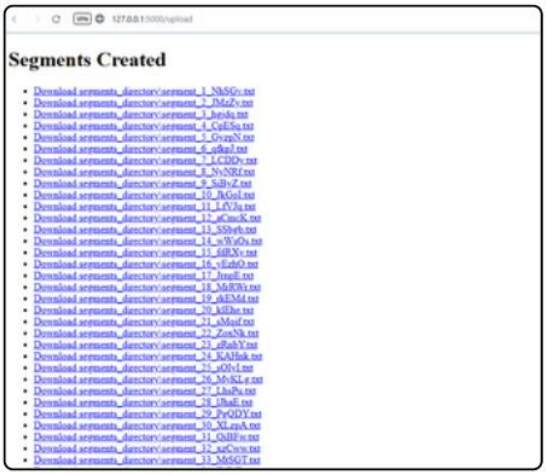
E. API & Service Layer

The emotional tone of the AI's reply is analyzed and categorized (neutral, encouraging, happy, corrective).

F. Frontend (Android .apk)

The user interface is delivered as an Android application built in Android Studio (Kotlin or Java). It features a file picker for uploads, screens for multi-factor login, and progress views that display segmentation and upload status. Upon completion, users can tap a button to download their reconstructed file. The app communicates securely with the backend via HTTPS using libraries like Retrofit or Volley.





VI. SECURITY ANALYSIS AND RESULTS

In our threat model, we assume an adversary may compromise one or more cloud storage providers, intercept data in transit, or attempt to breach the central server that holds the fragment map. To counter these threats, each file is split into n encrypted fragments and stored on separate cloud platforms. If an attacker gains access to any subset smaller than n, they obtain only meaningless ciphertext. Moreover, each fragment is encrypted with a unique content-encryption key (CEK) using either AES-256 or ChaCha20-Poly1305. These algorithms provide strong authenticated encryption, ensuring that fragment data cannot be modified or decrypted without the correct CEK.

The master key that protects the mapping of fragment IDs to cloud object locations and their CEKs is itself encrypted and kept in a secure vault on the central server. Users never see this map, and retrieving it requires successful two-step verification—such as an OTP, location check, or logic-based challenge. All client-to-server and server-to-cloud communications occur over HTTPS, thwarting man-in-the-middle and replay attacks. By isolating CEKs per fragment and hiding storage details behind an encrypted map, the system ensures that even if one component is compromised, attackers cannot reconstruct the original file or infer sensitive metadata.

VII. EVALUATION

The proposed file fragmentation and cloud distribution system has been successfully tested with various text files of different sizes, verifying the accuracy of the file splitting, naming, and reassembly logic. Currently, the project runs locally after being temporarily hosted on PythonAnywhere, with future plans to rehost for broader access. While the feature for storing fragments on multiple cloud services like AWS, Google Drive, and MEGA is under development, the current implementation handles directory-wise storage to simulate distribution. Encryption functionality, including AES and ChaCha20, is still under progress and will be integrated in the next phase to enhance security. User login with two-step verification, such as OTP and location-based checks, is planned to provide secure access during file retrieval. Although no major limitations have been identified at this stage, upcoming testing on cloud services and encryption modules will provide deeper insights into performance and scalability. This discussion highlights that while the foundational system works as intended, completing the encryption and cloud deployment will be key to fully realizing the project's security goals.

VIII. CONCLUSION

This project presents a secure and innovative approach to data protection by splitting files into encrypted fragments and distributing them across multiple cloud platforms. It minimizes the risk of data breaches by ensuring that even if one cloud is compromised, only a partial, unusable fragment is exposed. The system automatically handles file segmentation, encryption (to be implemented), and user

access through secure twostep authentication. Although currently running on a local host with cloud integration and encryption features pending, the project has proven its effectiveness in initial testing. In future work, the focus will be on completing and optimizing the encryption process using algorithms like AES and ChaCha20, enabling full cloud integration with services like AWS, Google Drive, and MEGA, and enhancing the user interface as an Android application. Further improvements will also include adding dynamic segment sizing, strengthening authentication methods, and addressing performance and scalability for large files

IX. REFERENCES

[1] Asritha Inakollu, S. Kranthi, and Jashua A, "A Novel Approach to Data Security in Cloud Storage using Erasure Coding and Re-Encryption," International Journal of Engineering Research & Technology (IJERT), vol. 11, no. 9, 2024

[2] B. Sengupta, A. Dixit, and S. Ruj, "Secure Cloud Storage With Data Dynamics Using Secure Network Coding Techniques," IEEE Transactions on Cloud Computing, vol. 10, no. 3, pp. 2090-2101, Jul.- Sept. 2022, doi: 10.1109/TCC.2020.3000342.

[3] G. Vasanthi, P. Chinnasamy, N. Kanagavalli, and M. Ramalingam, "Secure Data Storage Using Erasure-Coding In Cloud Environment," in 2021 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2021, pp. 1-4, doi: 10.1109/ICCCI50826.2021.9402639.

[4] Arfatul Mowla Shuvo, Md. Salauddin Amin, Promila Haque,"Storage Efficient Data Security Model for Distributed Cloud Storage" in 2020 IEEE 8th R10 Humanitarian Technology Conference (R10-HTC) | 978-1-7281-1110-0/20/2020 IEEE | DOI: 10.1109/R10-HTC49770.2020.9356962

[5] W. Shi, T. Liu, and M. Huang, "Design of File Multi-Cloud Secure Storage System Based on Web and Erasure Code," in 2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 2020, pp. 208-211, doi: 10.1109/ICSESS49938.2020.9237703.

[6] Vijay Kumar, "Brief Review on Cloud Computing", International Journal of Computer Science and Mobile Computing, vol. 5, September 2016.

[7] Victor Chang and Muthu Ramachandran, "Towards achievingDataSecurity with the Cloud Computing Adoption Framework", IEEE Transactions on Services Computing, Vol: 9, October 2015.

[8] Chandan Prakash and Surajit Dasgupta , "Cloud computing security analysis: Challenges and possible solutions", 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), November 2016.