# REPORT

# Distributed File Storage System

RAMAVATH CHINNA RAYUDU [22CSB0A45]

CH.HRUSHIK REDDY [22CSB0C13]

M.VAMSHI [22CSB0C21]

## Introduction

In today's digital world, securing sensitive data during storage and transmission is a critical challenge. Traditional file storage systems are vulnerable to **unauthorized access, data breaches, and tampering**, making it essential to implement a **secured distributed file storage** mechanism. The **Distributed Secured File Storage System** addresses these concerns by **ensuring confidentiality, integrity, and authentication** while managing data across multiple storage nodes.

**Problem Addressed**

The primary challenge in distributed file storage is **securing data against unauthorized access, data loss, and integrity violations**. Many conventional cloud storage solutions store files in centralized locations, making them **prone to attacks such as data breaches, insider threats, and ransomware attacks**. Moreover, managing **encryption keys** securely remains a concern, as **compromised keys** could allow attackers to decrypt sensitive information.

**Importance of the Problem**

With the increasing adoption of cloud-based services and distributed storage, **data privacy and security concerns** have grown significantly. Enterprises, government organizations, and individuals require **reliable, tamper-proof, and privacy-preserving storage** mechanisms to ensure that **confidential files remain protected** even when stored across multiple nodes. Additionally, **ensuring data integrity** is crucial, as any alteration in file content can lead to **security vulnerabilities, financial losses, or data corruption**.

**Existing Solutions and Their Limitations**

Several existing solutions attempt to secure distributed file storage using **cloud encryption services, blockchain-based storage, and decentralized networks**. However, they come with **limitations**, such as:

* **Static encryption keys** – If compromised, they expose the entire dataset.
* **Lack of dynamic key rotation** – Keys remain valid for long periods, increasing risks.
* **Limited integrity checks** – Many systems do not provide **end-to-end file verification** mechanisms.

**Key Contributions of Our Project**

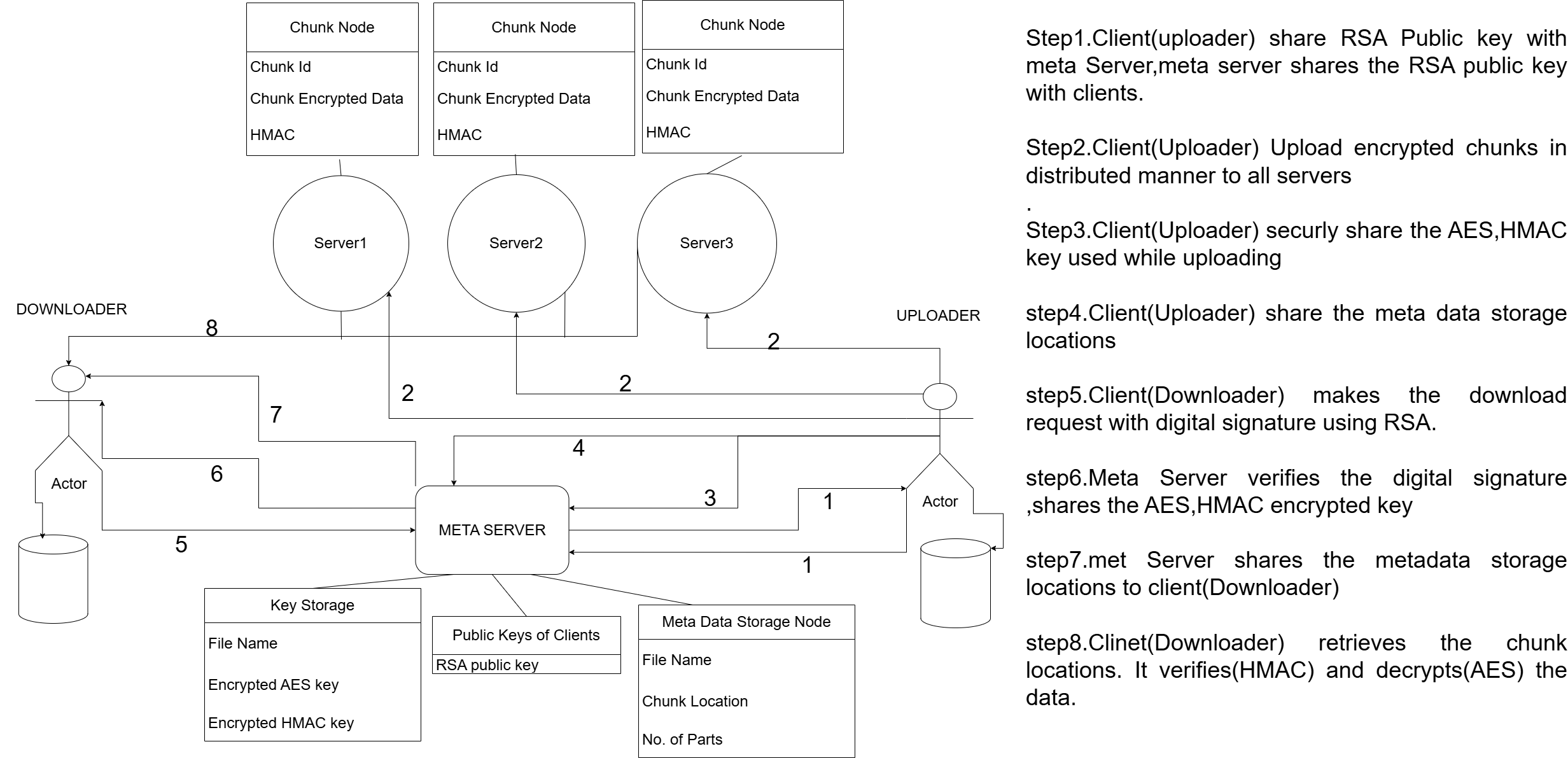
Our **Distributed Secured File Storage System** overcomes these challenges by incorporating:

1. **End-to-End Encryption**: Files are encrypted using **AES-256 encryption** before storage, ensuring data remains unreadable even if intercepted.
2. **HMAC-Based Integrity Verification**: Each file chunk is **authenticated using HMAC (Hash-based Message Authentication Code)**, ensuring **tamper detection** and verifying that data is not altered.
3. **Client Authentication via RSA Signatures**: Clients must **digitally sign** their requests, which the metadata server verifies before granting file access, preventing impersonation attacks.
4. **Distributed Storage for Scalability**: Files are divided into **chunks** and distributed across multiple **storage nodes(using Round Robin Technique)**, ensuring **fault tolerance and availability**.

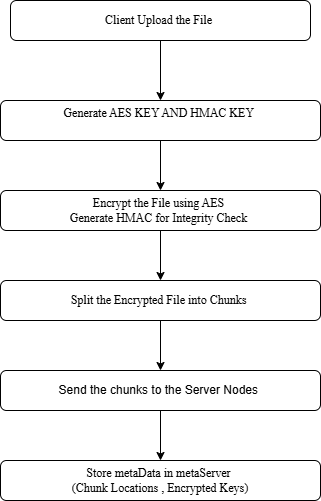
## Objectives

* **Ensure Integrity, and Authentication** by using **AES encryption, HMAC verification, and RSA-based authentication**.
* Dividing the File into chunks and storing them separately to avoid **fault tolerance**

## Project Flow Diagram



**Client Upload Flow Chart**

****

**Client Download Flow Chart**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Implementation of the Distributed Secured File Storage System :**

The **Distributed Secured File Storage System** is designed to store files across multiple servers securely, ensuring **confidentiality, integrity, and authentication** using advanced cryptographic techniques. Below is the step-by-step breakdown of its implementation.

**1. Implementation Steps**

**Step 1: Setting Up RSA Keys for Client & Metadata Server**

Each client and the metadata server must generate **RSA key pairs** for secure communication.

**Client-Side Key Generation**

* Generate a **2048-bit RSA key pair** (client\_private.bin & client\_public.pem).
* The public key is sent to the metadata server for **secure key exchange**.

**Metadata Server Key Generation**

* Generates its own **RSA key pair** (metadata\_private.bin & metadata\_public.pem).
* Clients use this **public key** of metadata Server to encrypt their AES & HMAC keys before sending them.

**Step 2: Secure File Upload Process**

1. **Client Generates AES & HMAC Keys for Encryption & Integrity**
   * **AES-256** key is generated dynamically **for each file**.
   * **HMAC key** (256-bit) is generated for **integrity verification**.
2. **Client Encrypts the File using AES-GCM**
   * AES-GCM ensures **confidentiality and authentication** (prevents data tampering).
   * The file is split into **chunks** (1024 bytes each).
3. **Compute HMAC for Each Chunk**
   * The **HMAC hash** is computed for each chunk before storing it on the servers.
4. **Upload File Chunks to Distributed Storage Nodes**
   * Each chunk is stored on a **different storage node** for **fault tolerance**.
   * Each chunk consists of **Encrypted data part** along with its **HMAC hash**
5. **Encrypt & Upload AES & HMAC Keys to Metadata Server**
   * The AES & HMAC keys are **encrypted using the Metadata Server’s public key**.
   * The metadata server stores these **encrypted keys** securely.
6. **Send File Metadata to Metadata Server**
   * Metadata includes:
     + Chunk locations on storage nodes.
     + Encrypted AES & HMAC keys.
7. **Delete AES & HMAC Keys from the Client**
   * The keys are removed from the client **to prevent reuse and compromise**.

**Step 3: Secure File Download Process**

1. **Client Requests File with RSA Authentication**
   * The client sends a **digitally signed request** (RSA-SHA256).
   * Metadata server verifies the **signature** before proceeding.
2. **Metadata Server Retrieves Encrypted Keys**
   * Decrypts the AES & HMAC keys with its private key.
   * Generates **new AES & HMAC keys** to re-encrypt the file.
3. **Metadata Server Re-Encrypts File Chunks**
   * Retrieves encrypted chunks from storage nodes.
   * **Decrypts using the old AES key** and re-encrypts with the **new AES key**.
   * Computes a **new HMAC for each chunk**.
   * Stores the **new encrypted chunks** back into storage nodes.
4. **Metadata Server Sends New Keys to the Client**
   * Encrypts the **new AES & HMAC keys** using the requesting client’s **public key**.
   * Sends them securely.
5. **Client Retrieves File Chunks & Verifies Integrity**
   * Downloads encrypted chunks from storage nodes.
   * Uses the AES key to **decrypt the file**.
   * Computes **HMAC** for each chunk and **compares it with the received HMAC**.
6. **Client Successfully Reassembles & Saves the File**
   * If all integrity checks pass, the client reconstructs the file.

### Algorithms

1. **RSA key Generation:** Used for Key exchanges
2. **AES Encryption:** Used for encrypting the file
3. **HMAC Algorithm**: Used for Integrity of the file , **Integrity Service** is achieved
4. **Digital Signature:** Done using RSA keys , **Authentication Service** is achieved
5. **SHA 512:** Used for signing the module at digital signature so that it can be verified
6. **Round Robin Algorithm:** Round robin technique is used to divide the file into chunks

**Conclusion**

The **Distributed Secured File Storage System** ensures **confidentiality, integrity, and authentication** in file storage using **AES encryption, HMAC verification, and RSA authentication**. By distributing encrypted file chunks across multiple servers, it enhances **security and fault tolerance**. The **dynamic key rotation** mechanism prevents unauthorized access even if keys are compromised. **HMAC ensures integrity**, and **RSA signatures authenticate clients securely**. This system is **scalable, resilient, and ideal for enterprises handling sensitive data**. With its robust security measures, it prevents **data breaches and tampering**, making it a reliable solution for **secure, distributed storage** in real-world applications.

## Learning Outcomes

- Understanding the principles of distributed file systems.

- Implementing data replication and fault tolerance mechanisms.

- Optimizing file retrieval using indexing strategies.

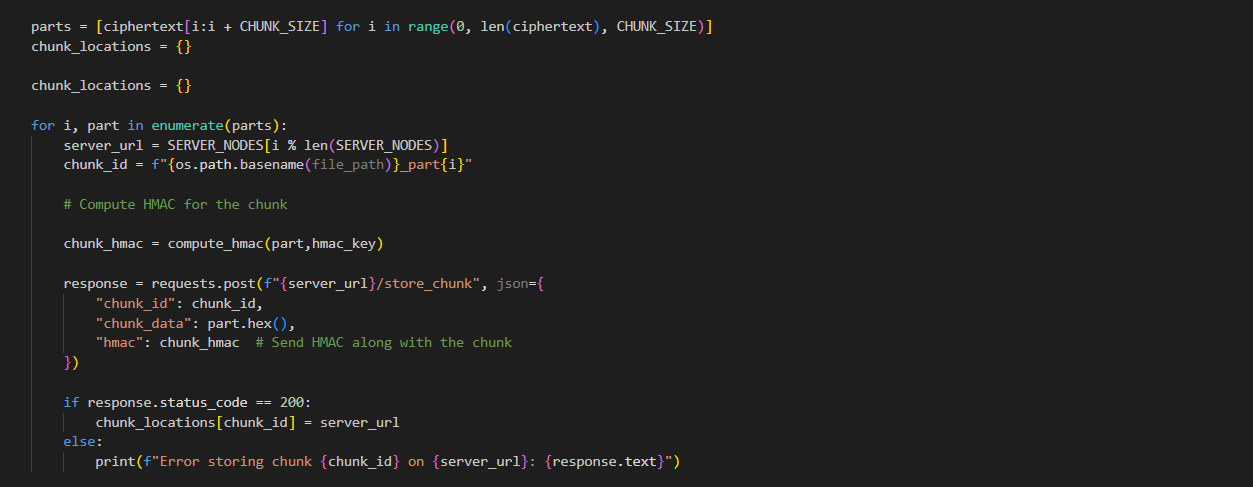
- Evaluating system performance and scalability.

-Applying Digital signature on a **HTTP API Get Request**

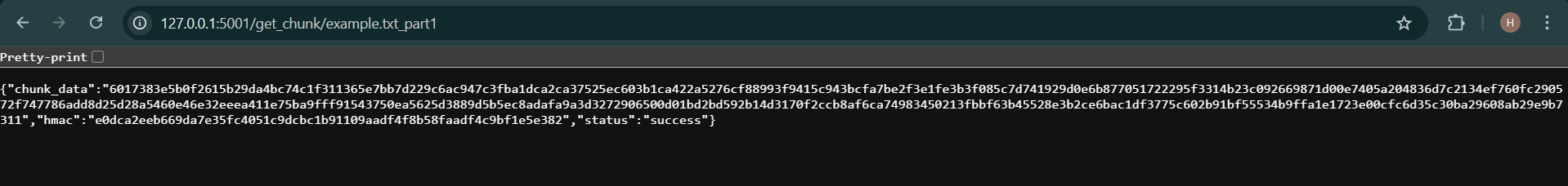
## KEY CODE SNIPPETS

**CHUNKING THE FILE AND STORING THEM AT STORAGE SERVERS**

**CODE :**

****

**Data at server Nodes :**

****

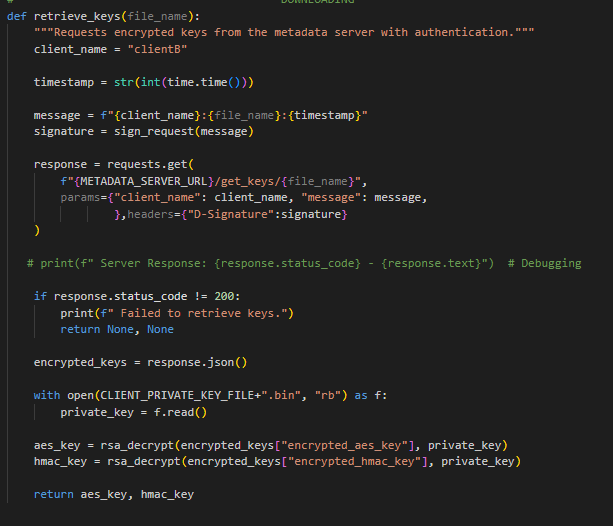
**UPLOADING THE ENCRYPTED KEYS AT THE METADATA**

**A computer screen shot of text

AI-generated content may be incorrect.**

**RETRIEVING THE KEYS**

**CLIENT SIDE REQUEST**

****

**DIGITAL SIGNATURE:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**META SERVER VERIFYING THE SIGNATURE :**

**A screen shot of a computer program

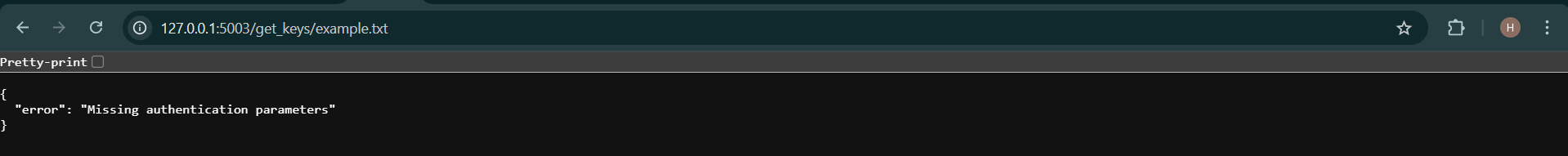
AI-generated content may be incorrect.**

**META SERVER GIVING THE KEYS BY VERIFYING THE SIGNATURE :**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**STORAGE OF KEYS AT META SERVER :**

****

**NOTE:** In above picture you can see that you cannot directly access the keys although you know the route where the keys are storing . You can only get the keys after authentication process only

**Verifying the integrity of chunks of data after retrieving from the Server Nodes**

**A computer screen shot of a program code

AI-generated content may be incorrect.**

## References