**A PEER-TO-PEER DISTRIBUTED ENCRYPTED FILE SYSTEM**

*CMSC 626 – PRINCIPLES OF COMPUTER SECURITY*

**CONTENTS**

1. ABSTRACT
2. PURPOSE OF A DISTRIBUTED FILE SYSTEM
3. WORKING OF A DISTRIBUTED FILE SYSTEM
4. ENCRYPTION
5. UML DIAGRAM
6. PSEUDOCODE
7. REFERENCES

**TEAM – 03**

Anjani Lingamallu BG39251 anjanil1@umbc.edu

Varunpal Gopidi DW89091 [vgopidi1@umbc.edu](mailto:vgopidi1@umbc.edu)

Seetaram Oruganti QL21570 [seetaro1@umbc.edu](mailto:seetaro1@umbc.edu)

Sai Teja Challa JA52979 [*schalla4@umbc.edu*](mailto:schalla4@umbc.edu)

Kavya Vellanki PI66332 [*kavyapv1@umbc.edu*](mailto:kavyapv1@umbc.edu)

***Under the guidance of,***

***Professor***

Gerald S. Tompkins

***Teaching Assistant***

Ran Liu

* + 1. **ABSTRACT:**

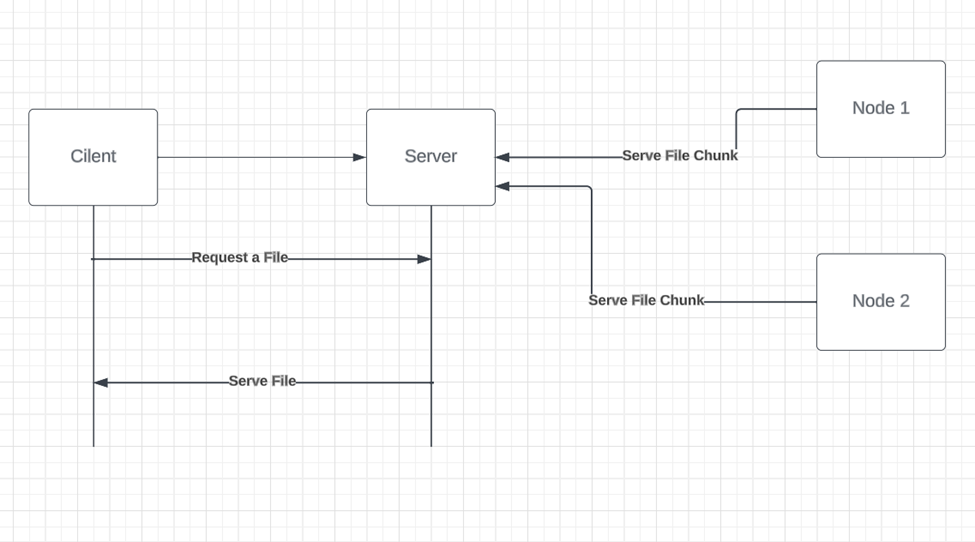
Modern digital communication needs advanced file-sharing services that prioritize privacy and integrity. This project presents a P2P distributed file system that allows users to store and transfer files on untrusted remote servers. This file system is intended to assure user confidentiality, data integrity, and real-time access to the most recent version of files. By adding advanced security mechanisms, permission-based user roles, encrypted communications, and an integrated log mechanism to detect malicious activity, we hope to overcome traditional P2P restrictions. This project's goal is to develop the next phase of the P2P file system that combines the efficiency of distributed networks with enhanced security features. Our solution focuses on user privacy, data integrity, and system stability, resulting in a platform that is streamlined, secure, and user-friendly for all file storage and sharing requirements.

* + 1. **MOTIVATION:**

A notable motivation for adopting P2P DFS is its inherent scalability, as it can easily scale without the need for reconfiguration. In a P2P distributed file system data is stored across multiple nodes, even if one node goes offline, the data can be accessed from remaining nodes. The files in this distributed system are encrypted using encryption techniques, hence these are always secure. When a client sends many requests, the jobs are distributed among multiple servers, and the system improves performance by reducing the workload on a single server. Thus, P2P DFS provides a robust environment for developing a secure file-sharing system.

* + 1. **WORKING OF P2P FILE SYSTEM:**

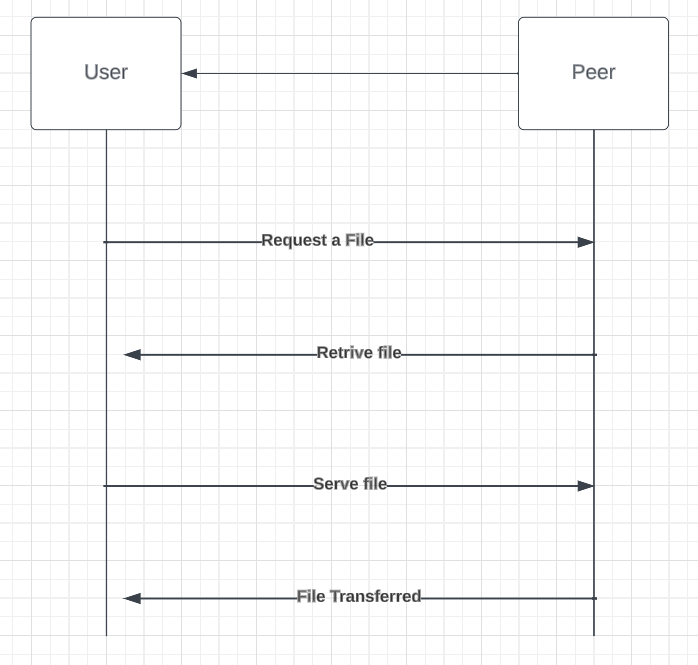
A distributed file system is a collection of systems also called peers connected to other peers to share files and resources. Here every peer acts as both client and server, sharing Their resources within the network. We should make sure all the users can see and are getting the most updated version of documents and are able to create, write, read, and delete files.



There will be special permissions and access restrictions to users for security purposes granting permissions to users by allowing or preventing a user from unauthorized access. A user can share a file with other users in the P2P Distributed File System network by adding it to their local file system. The user's P2P Distributed File System client software breaks down the file into smaller segments and encrypts them as a security measure against unauthorized access. Afterward, the client program disperses these file segments among different nodes within the network. Other nodes on the network can request specific file portions from these distributed nodes, enabling them to reconstruct the complete file on their respective systems. Assume there are 2 systems trying to share a file stored among distributed systems. Here, the client sends a request to peers in the network. Later the peer checks if they have a copy of the file requested by the client. If it does it acknowledges the request from the client and builds up a channel to share a file. if the peers have no copy to share the whole file, they need to request the bits of the file across the multiple nodes in the network. It receives all the bits, and the file is reconstructed before it is sent to the client. The client can perform reading and writing operations once it receives the file and all the changes saved will be the new version of the file.

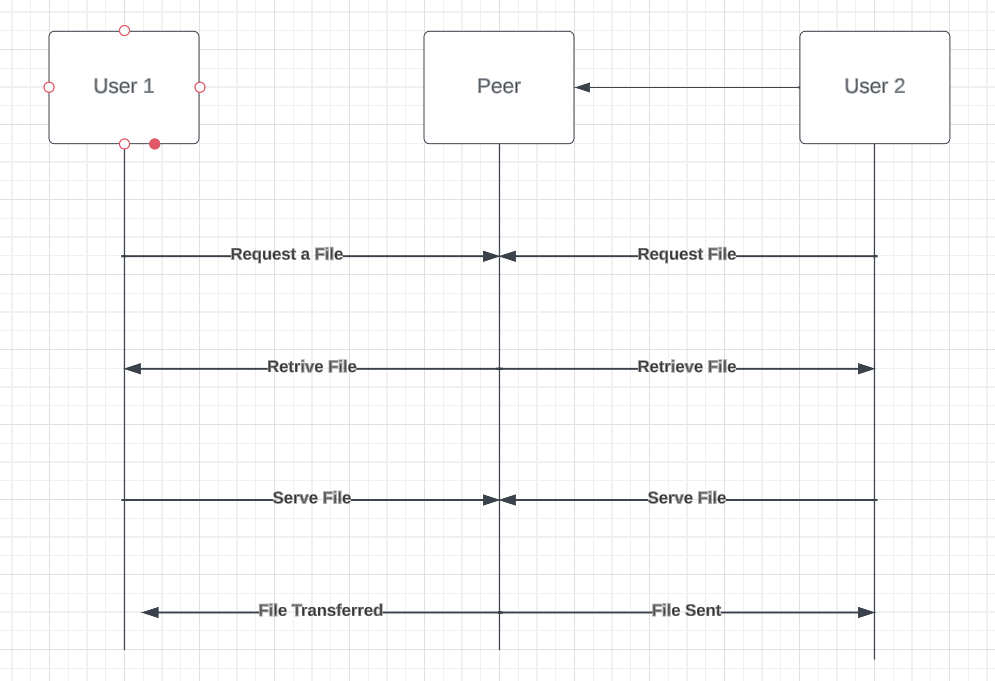
**Single Request Flow:**

In this scenario, a user initiates a request within a network, seeking access to a specific file or the execution of a particular file-related operation. The user's request is directed to a peer within the network. This peer, upon receiving the request, checks whether it contains the requested file. If it does not, the peer retrieves the file from another peer within the network. This file retrieval process may involve a certain amount of time. Once the peer successfully obtains the file, it promptly transmits it back to the user, thereby fulfilling the initial request. This completion of the request cycle allows the user to proceed with subsequent requests, whether they entail accessing different files or performing diverse operations. This intricate flow of information highlights the collaborative nature of peer-to-peer networks, where seamless file sharing and retrieval are facilitated among users and their interconnected peers.



**Multiple request flow:**

In the multiple-request flow scenario, multiple users can simultaneously request access to files or perform file-related operations within a peer-to-peer network. This concurrent influx of requests results in a complex interplay of interactions, necessitating the involvement of multiple peers in processing these demands and delivering the requested data. Ensuring a fair and efficient handling of these requests and secure and reliable data transfers is crucial. To achieve this, our approach involves implementing multi-threading techniques in the P2P Distributed file system. Peers must manage these requests, ensuring equitable treatment and timely file delivery. In this context, "serving" denotes a peer's action of fulfilling user requests, whether it involves retrieving files for reading purposes or performing modifications on files as requested by users. This meticulous process guarantees an optimal and responsive user experience within the dynamic framework of the peer-to-peer network.



* + 1. **ENCRYPTION:**

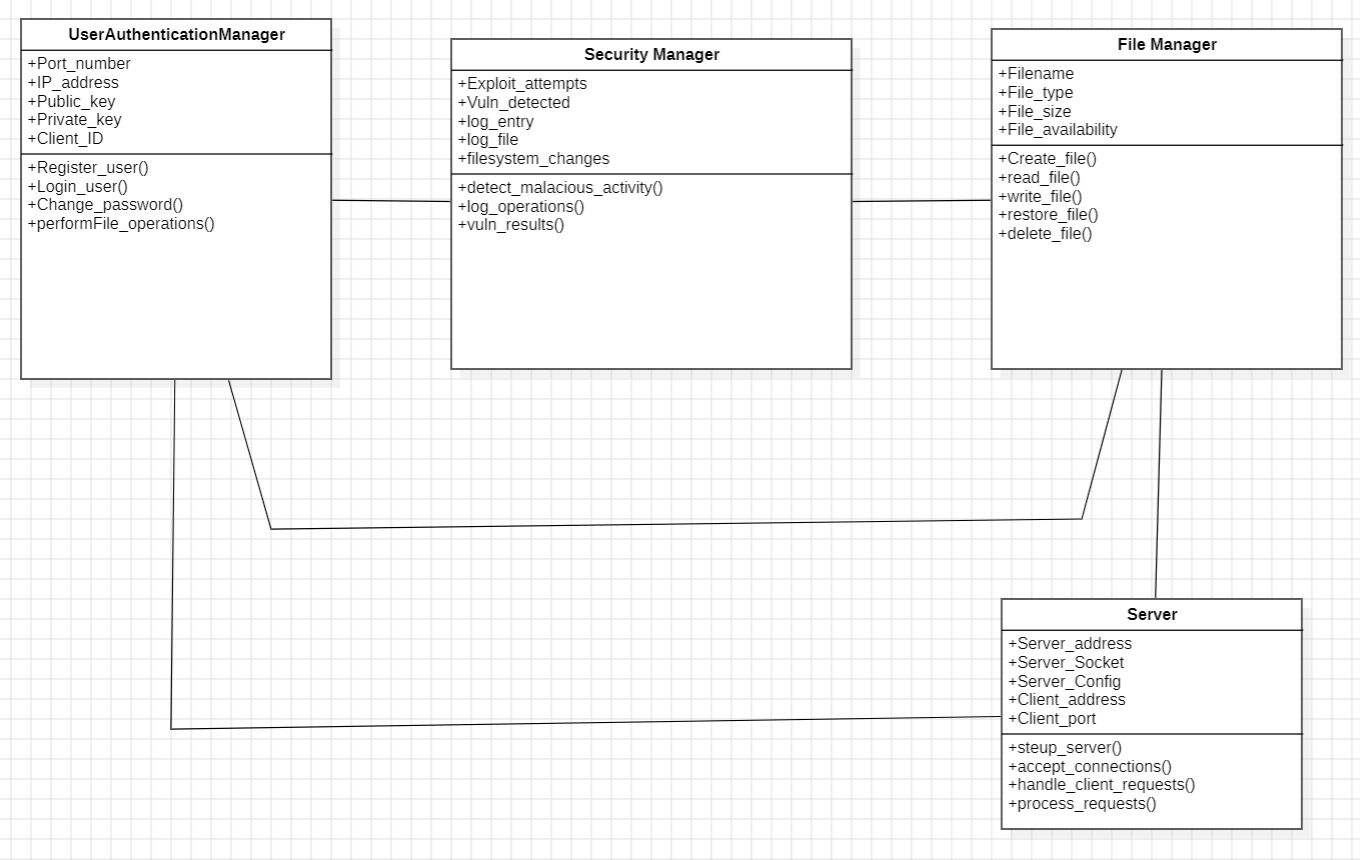
Encryption is to be implemented to secure the data while communicating or file sharing in the file system. We will be implementing the RSA algorithm as it is asymmetric and through that, we could achieve both Confidentiality and Integrity with the help of digital signatures. The RSA algorithm would be the best choice for implementing a basic encrypted file system.

PERMISSION MANAGEMENT SYSTEM:

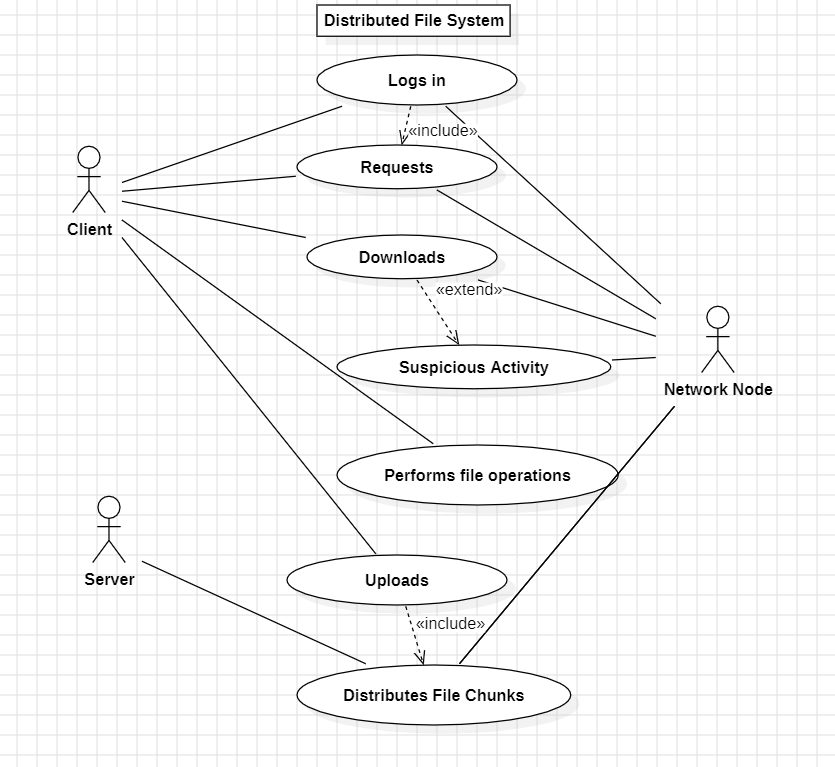
The permissions throughout this file system is done through role-based access system (RBAC). In RBAC, permissions are grouped into roles, and users are assigned to specific roles based on their job responsibilities. First, the roles are identified based on user responsibilities, such as admins, regular users, or guests. Specific permissions, like read, write, delete, or create, are then defined, and grouped according to these roles. When a particular user wants to access the files, they will send a request for a file which is encrypted. The system will then authenticate the user based on the role and its corresponding permissions such as read/write/delete depending upon the role. This takes place during the login process where the system authenticates using the username and password. The authorization process takes place once the authentication is done which can be implemented by checking the user’s permissions against access control model. Once this is done the system then proceeds to implement the necessary actions requested by the user.

**UML DIAGRAMS**

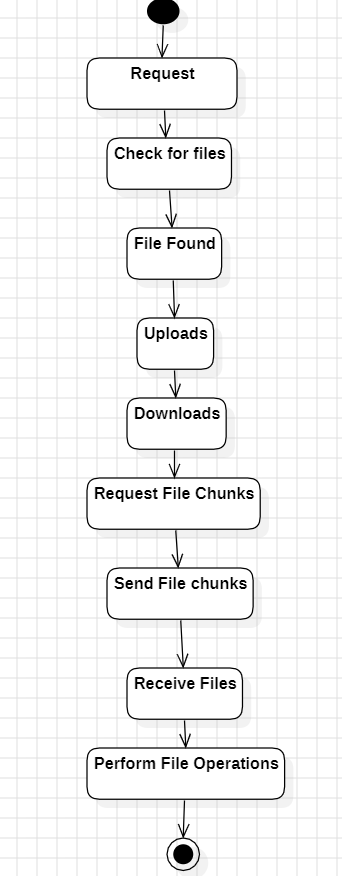
**Class diagrams:**



**Use Case Diagram:**



**Activity Diagram:**



* + 1. **PSEUDOCODE:**

**Import Required Libraries:**

Include necessary libraries and modules for encryption, networking, and database operations.

**Define System Configuration:**

Define replica server addresses and communication protocols for data synchronization.

**User Authentication and Management:**

register\_user(username, password):

Take username and password input

Hash the password

Check if username exists in the database

If yes, print error message

Else:

Insert user into the database

Generate RSA key pair

Insert public and private keys into the database

login\_user(username, password):

Take username and password input

Hash the password

Verify username and password

Retrieve public and private keys for the user from the database

change\_password(username, old\_password, new\_password):

Take username, old password, and new password input

Verify username and old password

Hash the new password and update it in the database

**File Operations and Security:**

create\_file(username, filename, content):

Check if user has create permission

If yes:

Create a new file

Encrypt file content with user's public key

Store encrypted data and metadata in the database

Replicate file creation to backups

Log transaction

read\_file(username, filename):

Retrieve encrypted file data from the database

Decrypt data using user's private key

Send decrypted data to the client

Log transaction

write\_file(username, filename, new\_content):

Check if a user has write permission

If yes:

Retrieve encrypted file data from the database

Decrypt data using user's private key

Update data with new content

Encrypt updated data with user's public key

Store encrypted updated data in the database

Replicate changes to backups

Log transaction

restore\_file(username, filename):

Check if a user has restore permission

If yes:

Retrieve file from the restore folder

Copy file to the main folder

Replicate changes to backups

Log transaction

delete\_file(username, filename):

Check if a user has deleted permission

If yes:

Move file to the restore folder

Replicate deletion to backups

Log transaction

**Security Measures:**

detect\_malicious\_activity():

Implement features to detect unauthorized file creations or deletions by monitoring file system changes.

Generate logs for unauthorized access/modifications and store them securely.

log\_operations(username, operation\_type, filename):

Incorporate a logging system to record every operation on the server, including authorized and unauthorized actions.

Implement log analysis algorithms to identify patterns indicative of attacks.

vulnerability\_testing():

Define potential vulnerabilities in your detection system.

Attempt to exploit these vulnerabilities using techniques from the first project to validate the effectiveness of your security measures.

**Main Server Logic:**

Setup server with defined configurations

Accept client connections and spawn threads for each client

Handle client requests by calling appropriate handler functions based on the received command

Implement a loop to continuously receive and process requests from clients".

* + 1. **REFERENCES:**

1. [Peer-to-peer file sharing - Wikipedia](https://en.wikipedia.org/wiki/Peer-to-peer_file_sharing)
2. [P2P (Peer To Peer) File Sharing - GeeksforGeeks](https://www.geeksforgeeks.org/p2p-peer-to-peer-file-sharing/)
3. [File Encryption and Decryption System Based on RSA Algorithm | IEEE Conference Publication | IEEE Xplore](https://ieeexplore.ieee.org/document/6086320)
4. [Lecture12.pdf (purdue.edu)](https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture12.pdf)
5. [RSA (cryptosystem) - Wikipedia](https://en.wikipedia.org/wiki/RSA_(cryptosystem))