**SECURELY ENCRYPTED DISTRIBUTED FILE SYSTEM**

**AUTHORS:**

Vishnu Vardhan Samba - IK67010

Parinitha Reddy Gaddam - KT16974

Pragadiswar Nanda Muralidharan - PO21974

Hima Poojitha Sai Sree Myla - QN08170

**INTRODUCTION:**

This Project is aimed to construct an Encrypted Distributed File System that enables the User(s) to perform certain operations such as Read, Creation, Deletion and Modification of the Data that is stored in the Server.

**FUNCTIONALITY:**

* **PRIMARY PHASE**

The chosen Programming Language for the Implementation is Python (Version 3.10.7).

The Initial Phase of Implementation includes 5 Python Scripts (.py) that are aimed individually towards one Communication Server and 4 nodes and respectively. All the scripts are aimed at Specific Functionalities with suitable Access Restrictions. The Communication server script is mainly used for node-to-node Interactions. The node Script is aimed towards performing all the necessary operations including CRUD operations on the file system, alongside providing the interface for user interaction. Node script can either be implemented in a single machine or on different machines in the same network. When implemented on different machines, the nodes will be sharing different resources of their respective machines, performing their respective operations and transmitting the results to all the other nodes.

* **COMMUNICATION SERVER**

No CRUD operations or operations on file system are performed on the Communication server. The Communication server only acts as a medium for the nodes to interact and helps in transmitting data from one node to another.

This information is maintained across the other Server(s) that are present in the System maybe even for redundancy.

One other main function of the Communication server is that in a case where one of the node is down or not active yet, it queues all the requests so that as soon as the node is up, it will update the node by passing on all the queued operations for that node to perform.

Any communication between nodes and Communication server and vice-versa are encrypted. “Pycryptodome” module will be used for this purpose. In that, AES encryption will be used as a method to encrypt the data. Which implies that all the communication between the nodes is encrypted. The concepts of Sockets and Multi-threading is implemented during connection with nodes and transmitting the data.

**NODE/USER SCRIPTS:**

In this implementation, 4 identical scripts are used specifically, one for each student, acting as 4 nodes.

Multiple operations are performed by the User scripts:

1. It is the main script that performs all the required CRUD operations.
2. It transmits the results of operation to all the other nodes with the help of Communication server.
3. It makes sure that result any operation performed by a legitimate node is broadcasted across all the nodes and all the nodes are updated accordingly.
4. It takes care of user permissions, which include allowing/blocking the users from accessing files.
5. It acts as an Interface for Users to provide input to perform operations.
6. It makes sure that all the files stored in each node is encrypted.
7. It sees to it that an unauthorized user will not be able to see or access the user specific unauthorized files.
8. Detects malicious file server.

Under the User/Node Script, The User Interface is enabled for Client Interaction to Server and obtain corresponding Response including Connection Request/Response to/from Server, File Operations to be performed, Server Response Post Operation Status (Success/ Failure). Whenever New file is Created and sent by the User, all Necessary Details must be ensured that is present including Name, Permissions.

To handle multiple requests, we plan to incorporate multi-Threading where in the In-Built python Library is utilized.

Whenever the Request has been seen from User, it will be Threaded as the Nodes will perform the operations concurrently. This Threading will enable multi-processing simultaneously.

The Socket package is utilized to handle the System. Suitable socket commands are used to opening Sockets, receive and Close sockets as well. This can be written as s.connect(‘ip’, ‘port’), s.recv(), s.bind(), etc.

**AES ENCRYPTION & DECRYPTION:**

The AES has been employed for securing the System. Keys are randomly generated at Client and used to encrypt the message sent to Storage Handler.

Key = get\_random\_bytes()

Ciphertext.tag = cipher.encrypt\_and\_digest(data)

File\_out = open(encrypted\_file)

File.out.write = for(x) in (cipher, tag)

During Decryption, the file is deciphered using AES.MODE\_EAX.

Cipher = AES.new(key, AES.MODE\_EAX).

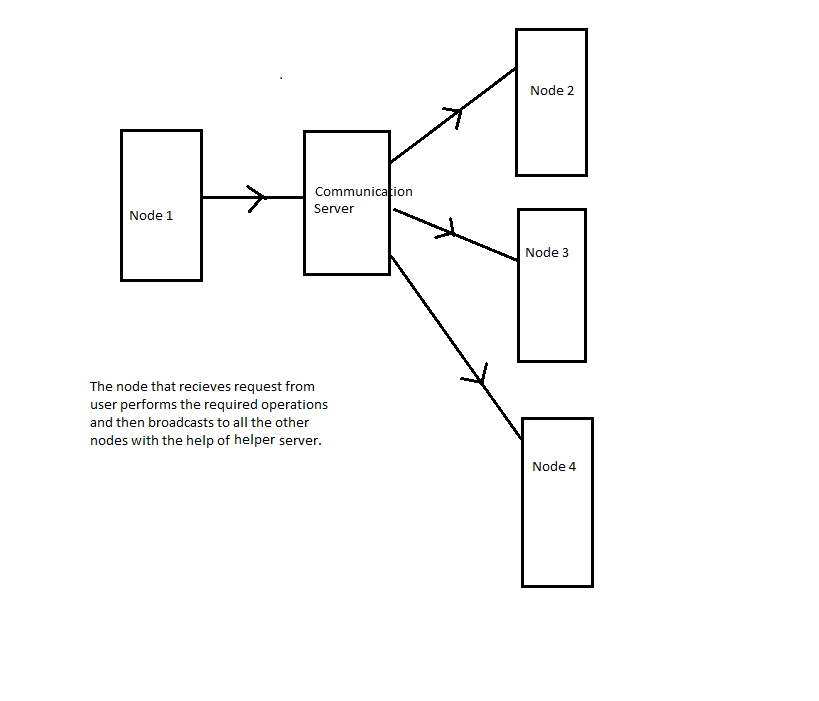
Data = cipher.decrypt\_and\_verify(ciphertext, dag).

**INTERACTION BETWEEN THE PEERS AND THE SERVER**Diagram

Description automatically generated

The above diagram depicts the connection between the peers and the Communication server briefly.

**SYNCHRONIZATION OF THE FILES BETWEEN THE SERVERS**

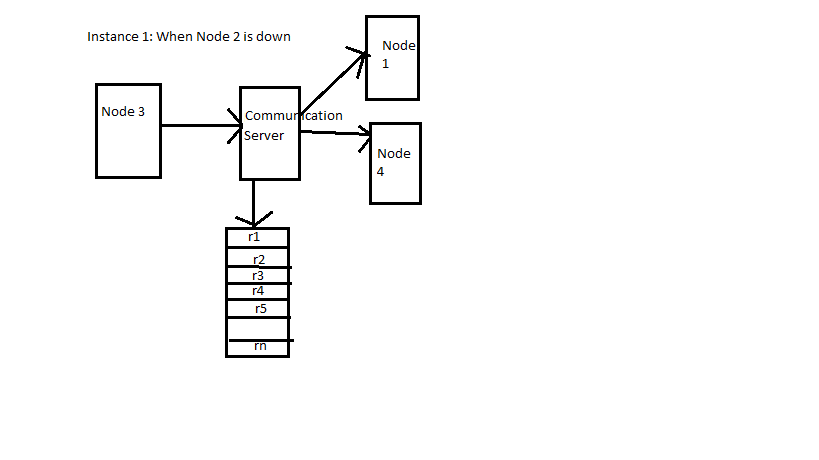


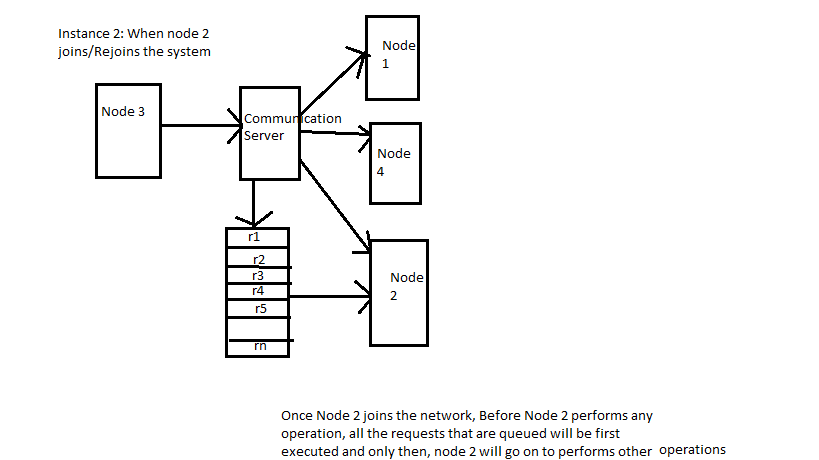
The Client will be able to add Permissions for the files. Any user that tries to perform an unauthorized action on a file will be unable to do so without necessary permissions. The permissions will be given to when the file is created.

**HOW THE FILE INFORMATION IS STORED WITH USER PERMISSIONS**



**WHAT HAPPENS WHEN ANY OF THE SERVER IS DOWN**





In the situation where one of the node/ server is down or in case where the node has not joined the network yet, the communication server maintains all the requests that are thus far implemented by all the other nodes in a queue. Once the node joins/rejoins the network, the first task that happens is that all the requests that are queued will be executed by newly joined node. Only after this is done, the node will accept requests from user and goes on to perform its operations.

**How are Permissions managed:**

When a file is created, permissions are given as input argument to the create function. In the backend, there are 4 lists that exist for each user: Read, Write, Delete and Restore.

For instance, a user, say Alice, has Read, Write and Delete permissions for a certain file xyz but doesn’t have Restore permission. In this case, the file xyz is appended to the lists Read, Write and Delete of user Alice but not to Restore list. Next, when Alice wants to perform any one of the other 4 operations, first the lists are checked for the file. If the required file exists in the appropriate list, that implies that Alice has the permission. If not, a message will be displayed that Permission is denied.

**How CRUD operations are Handled by Nodes:**

**CREATE:**

When create operation is performed, say at node2, the following operations happen:

1. Permissions of the file are set
2. File is created in node 2 in assigned path
3. The file that is created is replicated in all the other nodes via communication server.

Create function has 2 arguments: File name, permission

When a node performs create action, internally it makes use of 3 functions:

1. Send\_request(user)
2. Get\_request()
3. Get\_queue\_request()

In the create part of send\_request(user) function, first it checks if the file already exists. If that is the case, an output message is displayed for user saying that the file exists. On the other hand, if the file does not exist, it is created using fo\_open() function. Also, according to the permission argument given with the create function, the file will be added to the Write, Read, Delete, Restore lists of all the other users. Later, it will be sent to communication Server to broadcast to all the nodes. One other way of execution of Create command is if the node receives the command from Get\_reques() or Get\_queue\_reques() functions. A node receives Create command from Get\_request() function when some other node gets the user input to create that file and that node is broadcasting it to all other nodes.

The command is received by get\_queue\_request() function when the current node was inactive and just came up.

**READ, WRITE, DELETE AND RESTORE Operations:**

When one if these commands are encountered by a node, first the node checks if the user accessing the node is authorized to perform the operation, by checking the respective user operation list for the file name. If the user has the access, the commands are executed in the node and is broadcasted, if required, to all other nodes.

All these operations make use of the same 3 functions: Send\_request(user), Get\_request(), Get\_queue\_request(). If the operation is user input to the node, it uses the function send\_request() to check for permissions, execute the functions and in case of all other operations except Read, broadcasted to all other nodes. If the operation(other than read) is first performed in some other node, all the rest of the nodes get the request that is broadcasted Get\_request() function. If a node is inactive previously and now active, it gets the requests through Get\_queue\_request()

The following are given as arguments for each Operation:

1. Read: File name
2. Write: File name, content to write
3. Delete: File name
4. Restore: File name

**HOW ARE CONCURRENT READ AND WRITE OPERATIONS HANDLED:**

As the concepts of multi-threading are used throughout the project to handle various operations, it is necessary to implement the concepts of locks to avoid racing condition and obtain accurate results. Here, the functions acquire() and release() of threading module are used to lock and release parts of code. To be more specific, when a thread performs an operation on a file, specifically read and write, acquire() function is used such that the particular file is locked for other threads and all the other threads that need access to that particular file have to wait. Once the write operation is done and lock is released, other threads can access that file to perform other operations.

