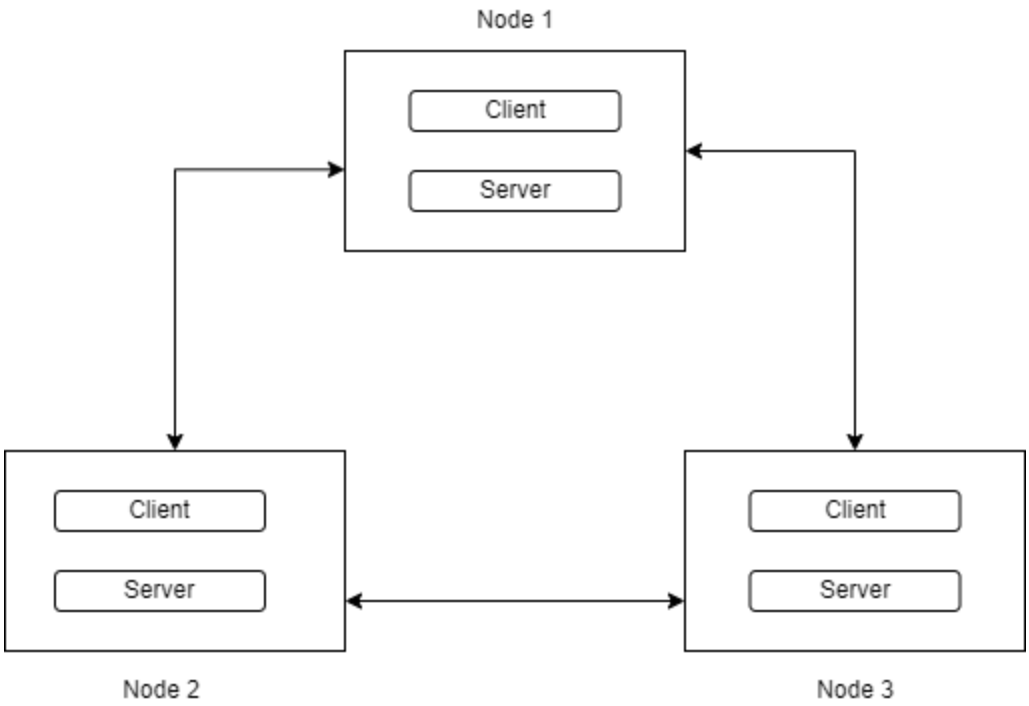
An Encrypted P2P File System

This is an encrypted peer-to-peer file system in a distributed setting that will allow the nodes in the network to store the data on other untrusted servers. We break down the architecture as a client-server model, where the client and server will be running on all the nodes in the network.

Initial Setup

Since this is a P2P, to simplify our efforts is considering maintaining the list of all the users part of the file system on all the nodes. This list will be used by the client and server to make further requests.



Client Interface

Our interface will be similar to that of a Linux terminal. The client program on execution will be ready to accept the supported functionalities. We plan to support the following features:

- Update the user's keys
- List all the user files
- Read a file
- Write to a file
- Grant read/write permissions to other users

To start with any of the features, the user first must create a private/public key pair. These keys will form the basis for the encryption being used in the file system. The public keys of all the users will be shared with others in the network.

File Creation

The design bases its idea on creating a unique RSA key for each file. This safely allows sharing the file with other users. When a user creates a file, the client will generate a random RSA private and public key to encrypt the data. After the keys are created, the client interface will allow writing data to the file. When the user has completed writing content, the data will be encrypted with the public key which was generated for the file.

Now the encrypted data will be replicated on the peers. Since the keys for the files are not shared with other users, they won't be able to access the file content.

Let's say, P is the private key and Q is the public key generate for the file F. Then the encrypted text will be given as $e = Encryption(F, Q)$. While to decrypt the user can simply do $F = Decryption(e, P)$.

File Sharing

When a user wants to share a file with others, we will have to also share the key which was used to encrypt the data. This key can be shared securely by encrypting with the public of the user with whom to share with. Due to this, only the recipient user can decrypt the shared key.

Consider an example, we have $P(F), Q(F)$ the private and public keys for the file F created by the Node_1. Now, Node_1 wants to grant read access to Node_2. In order to read, Node_2 will require $P(F)$. To send it over to Node_2, the Node_1 encrypts it with $Q(Node\_2)$. Therefore, the read key for Node_2 will be given by $readKey = Encrypt(P(F), Q(Node\_2)$. This key will be shared by the Node_1 to Node_2.

Similarly, if Node_1 wants to grant write access to Node_3 we will share the public key of the file as $writeKey = Encrypt(Q(F), Q(Node\_3)$. The other users who don't have $Q(F)$ won't be able to update the file, as the key to encrypt will be missing.

Key Revocation

If a user requests to change the keys then all the read and write keys created with it will have to be recomputed and updated. Once these keys are updated the user can access all the content with the same privileges.

Consistency

Any user will see the file content up to date to the time when the request was processed. If any user modifies the file, the others will have to reload the file to see the changes. We currently don't plan to support concurrent writes.

Replication

To simplify the design, we plan to replicate the file on all the nodes in the file system.

Design

Network

- For simplicity, we start with a static configuration.
- Each node will have a list of peers and their IPs.
- At the start, each node will have to share its public keys with all other nodes in the network.

Create File

- The node which creates the file will be called the owner of the file.
- The owner node will be responsible for creating and maintaining public and private keys for the file.
- The user will save the file with a file name, and also create a unique file id.
- After the file is saved, the owner node will encrypt the file along with its name and sign off with its private key (owner node's). After signing off the owner node will replicate the file and its associated file ID to all the nodes.
- Each node upon receiving the replicated copy will save it to its preferred location and make an entry into its DB.

Owner	File_ID	Path_To_File

Grant Permission

- The only owner node can grant permission.
- The client interface on the peer node will have the functionality to do so.
- The owner node will also maintain a ledger to keep track of nodes that have been given permission.

Read Permission

- The owner node will share the private key of the file to be shared with the other node to grant the read permission.
- To do so, the owner node will encrypt the private key of the file with the other node's public key. This way any other nodes won't be able to access the key.
- The other node upon receiving the permission will make an entry of the shared file with itself for future reference.

Owner	Permission - R	File_ID	Private_Key_Of_File	<NO PUBLIC KEY>

Write Permission

- In this case, the owner node will share both private and public keys with the other node. The table entry on the other node will look like

Owner	Permission - RW	File_ID	Private_Key_Of_File	Public_Key_Of_File

- The private key will be used to read the file content.
- The public key will be used by the other node to encrypt back the file and share the updated encrypted file back to the owner node which will later take care of updating it on all other nodes. (More details in Write File section)

List Files

- Any node can only see the files owned by it or shared with them.
- To make things simple, each node will maintain the details about files owned by them and shared with them (as shown in the Grant Permissions section).
- The file name would be encrypted and the nodes will have to use the file's private key to decrypt it. This follows the same mechanism as used to read the file.

Read File

- To read a file, the node must be either the owner or have permission to read.
- The private key of the file needs to be used to decrypt the content.

Write File

- Similarly to write a file, the node must be either the owner or have permission to write.
- Here the node needs both private and public keys to make a write.
- The public key will be used to decrypt the existing content while the public key will be used to encrypt the file.

Security

- The main security challenge is that no node that doesn't have permission can update the content of the file in the network.
- To solve this, we intend the owner node will always be responsible for replicating the content.
- The other nodes that have permission and wish to make any change, will share the updated content with the owner node (after editing the file). The owner then will replicate it to other nodes. The other nodes will have an existing entry for the file and will check if it was sent by the owner node with the help of a digital signature. (Note that each node has the public key of all other nodes)

Concurrent Writes

- Since a file can be shared with multiple other nodes we need to resolve the concurrent write problem.
- For the project, we plan to keep it at a simple level.
- Any node that intends to write will first have to take a lock over the file. This lock will be maintained by the owner node.
- Upon completion of the write, the lock will be released.

Delete File

- Only the owner node can delete the file.
- Each node upon receiving will check for the digital signature to validate the request.
- If successful then the node will delete the file and all other corresponding entries maintained by it

Key Revocation

- Any node in the network will be able to change its RSA keys.
- The new public key will then be sent to all other nodes.
- Since, the files have their own key we don’t need to worry about it. The owner node will be able to access the files as before.

Revoke File Permission

- This is the most tedious functionality.
- To revoke any permission we need to regenerate RSA keys for the file. These new keys will then have to be shared with other nodes that have been given access.