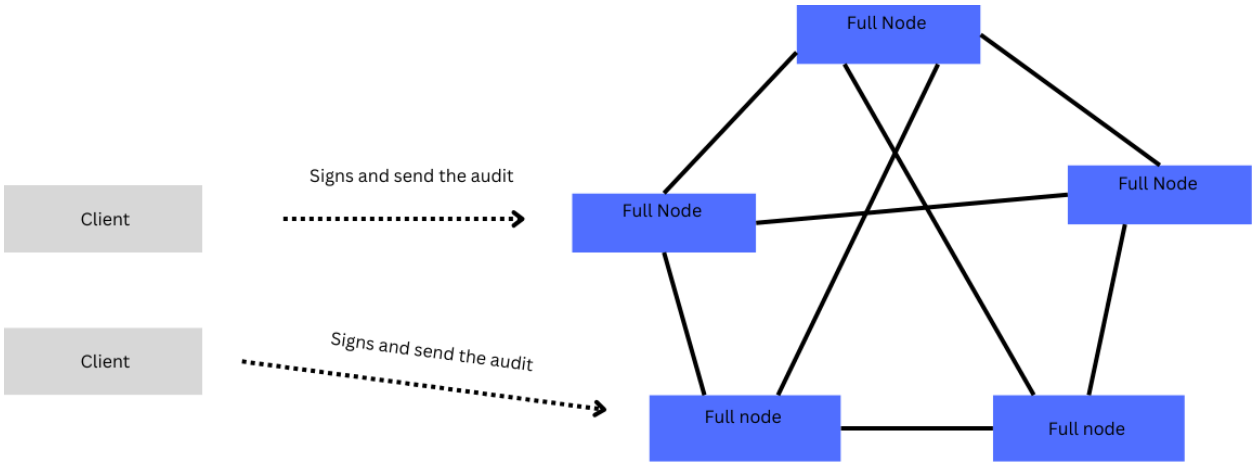
BlockChain Implementation



Block chain type - **Private**

Consensus - **PoA**
This type of blockchain secures itself by using identity. Every node is pre-selected to verify audits, and the identity of these nodes are known. In case of any wrongdoing, a node can be held accountable for its actions. Because there is a smaller set of nodes and users required to come to consensus, this blockchain can be fast and easily scalable.

Client Process

1. Format the audit
2. Sign it using digital signature
3. Send it to the Full node process

Full node Process

- Every Full node Checks the audit request - Validates the audit, adds the request to mempool and whispers the request to the neighbors.
- Every node has the mempool which has all the unprocessed requests. Mempool is randomly ordered unprocessed requests not necessarily a queue (linked list?).
- Every audit's signature is verified by every node.
- Consensus based validator is selected among the nodes, e.g. we elect a single leader who proposes blocks or some other consensus algorithm
- Block is proposed by the validator node

- After block acceptance based on the consensus the requests in blocks are removed from mempool and block is confirmed.

Need to analyze:

- **Consensus algorithm to find block proposer**
- **How do nodes recover if they fail ?**
- **How does each node verify the signature of the audit?**

Question section:

1. **How does every node verify the correctness of the audit?**
    a. Every node verifies the digital signature by the client.

Project Implementation Plan

To encourage parallel development and promote architectural diversity, the project will be divided across 8 independent teams, each responsible for delivering a complete and self-contained blockchain module. Specifically, each team will:

1. Develop a client application that creates audit records (creates the FileAudit proto), applies a digital signature, and submits them to the network (rpc call to some Full Node).
2. Implement a full node capable of handling mempool operations (storing unprocessed requests), validating audits (checking signature against public key), participating in block creation (voting on blocks), and maintaining blockchain state (saving blocks to disk). *** A shared consensus mechanism (PoA)*
3. Design and simulate a node recovery mechanism, demonstrating how the node detects and recovers from faults or crashes.
4. Utilize any programming language or system architecture of your choice, promoting innovation and flexibility.
5. By the end of the development cycle, these modules will be integrated to form a multi-node private blockchain system, enabling evaluation of inter-node communication, consensus behavior, and fault resilience across heterogeneous implementations

Common.proto

```
syntax = "proto3";

package common;


enum AccessType {
      UNKNOWN = 0;
      READ = 1;
      WRITE = 2;
      UPDATE = 3;
      DELETE = 4;
}

message FileInfo {
      string file_id = 1;    // e.g. inode on Linux/Mac (stat command)
      string file_name = 2;
}

message UserInfo {
      string user_id = 1;    // e.g. id --user on Linux
      string user_name = 2;
}

message FileAudit {
      string req_id = 1;     // incase of async
      FileInfo file_info = 2;
      UserInfo user_info = 3;
      AccessType access_type = 4;
      int64 timestamp = 5;
      string signature = 6;    // To verify the signature
      string public_key = 7;   // To verify the signature
}
```

Public key: rsa keys generated using `ssh-keygen`
Signature: private rsa key + sha256 hash + PKCS1v15 padding (default padding of openssl)

file_audit.proto

```
syntax = "proto3";

import "common.proto";

package common;


message BlockHeader {
      string block_hash = 1;
      uint64 block_number = 2;
      int64 timestamp = 3;
      string previous_block_hash = 4;
      string merkle_root = 5;
      repeated string merkle_proof = 6;
      repeated string audit_hashes =  7; // optional for debugging and
verification
}

message FileAuditResponse {
      string req_id = 1 ;                // incase of async
      string blockchain_tx_hash = 2 ;
      BlockHeader block_header = 3 ;
      string status = 4 ;  // success, failure
      string error_message = 5; // If its failure
}

service FileAuditService {
      rpc SubmitAudit ( common.FileAudit ) returns ( FileAuditResponse );
}
```

block_chain.proto

```
syntax = "proto3";

package blockchain;

import "common.proto";


message WhisperResponse {
      string status = 1; // success, failure
      string error = 2 ; // error message incase of failure
}

message Block {
      string block_hash = 1;
      string previous_block_hash = 2;
      string merkle_root = 3;
      int32 block_number = 4;
      int64 timestamp = 5;
      string proposer_id = 6;
      repeated FileAudit audits = 7;
}

message BlockProposal {
      Block block = 1;
      string proposer_id = 2;
      int64 timestamp = 3;
}

message Vote {
      string block_id = 1;
      string validator_id = 2;
      bool approve = 3;
      string signature = 4;
      int64 timestamp;
}

message  BlockVoteResponse {
      bool success = 1;    // Success or Failure
      string message = 2; // Error message
}
```

```
service BlockChainService {
      rpc WhisperAuditRequest ( common.FileAuditRequest ) returns (
WhisperResponse );
      rpc ProposeBlock( BlockProposal ) returns (BlockVoteResponse);
      rpc VoteOnBlock ( Vote ) returns (BlockVoteResponse);

      //rpc FinalizeBlock ();
}
```