DAO Voting Research Implementation Guide

A Step-by-Step Implementation Guide for Privacy-Preserving DAO Voting System

Phase 1: Foundation Setup (Weeks 1-2)

1.1 Development Environment Setup

Required Tools:

- Node.js (v18+ recommended)
- VS Code with Solidity extensions
- Git for version control
- MetaMask browser extension

Project Structure:

```
dao-voting-research/
├── contracts/          # Smart contracts
├── scripts/            # Deployment scripts
├── test/               # Test files
├── frontend/           # React.js frontend (later)
├── zk-circuits/        # ZKP circuits (later)
├── docs/               # Documentation
└── package.json
```

1.2 Basic Hardhat Configuration

Your current setup is good! Enhance your `hardhat.config.js`:

```javascript
```

```
require("@nomicfoundation/hardhat-toolbox");
require("dotenv").config();

module.exports = {
  solidity: {
    version: "0.8.20",
    settings: {
      optimizer: {
        enabled: true,
        runs: 200
      }
    }
  },
  networks: {
    hardhat: {
      chainId: 1337
    },
    mumbai: {
      url: "https://rpc-mumbai.maticvigil.com",
      accounts: [process.env.PRIVATE_KEY || ""]
    }
  }
};
```

Phase 2: Enhanced Smart Contract Development (Weeks 3-6)

2.1 Current Contract Analysis

Your basic contract is a good start! Here are the improvements needed:

Issues to Address:

- No vote tracking per user (allows multiple votes)
- Missing weighted voting mechanism
- No token-based voting power
- No time-based proposal lifecycle

2.2 Enhanced DAO Contract

solidity

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

contract EnhancedDAOVoting is Ownable, ReentrancyGuard {
    IERC20 public governanceToken;

    struct Proposal {
        uint256 id;
        string title;
        string description;
        uint256 startTime;
        uint256 endTime;
        uint256 yesVotes;
        uint256 noVotes;
        bool executed;
        mapping(address => bool) hasVoted;
        mapping(address => uint256) voteWeight;
    }

    struct Voter {
        bool isRegistered;
        uint256 reputationScore;
        uint256 registrationTime;
    }

    mapping(uint256 => Proposal) public proposals;
    mapping(address => Voter) public voters;
    uint256 public proposalCount;
    uint256 public constant VOTING_PERIOD = 7 days;

    event ProposalCreated(uint256 indexed proposalId, string title);
    event VoteCast(uint256 indexed proposalId, address indexed voter, bool support, uint256 weight);
    event VoterRegistered(address indexed voter);

    constructor(address _governanceToken) {
        governanceToken = IERC20(_governanceToken);
    }

    function registerVoter(address _voter, uint256 _reputationScore)
```

```
        external onlyOwner {
        voters[_voter] = Voter({
            isRegistered: true,
            reputationScore: _reputationScore,
            registrationTime: block.timestamp
        });
        emit VoterRegistered(_voter);
    }

    function createProposal(string memory _title, string memory _description)
        external onlyOwner returns (uint256) {
        proposalCount++;
        Proposal storage newProposal = proposals[proposalCount];
        newProposal.id = proposalCount;
        newProposal.title = _title;
        newProposal.description = _description;
        newProposal.startTime = block.timestamp;
        newProposal.endTime = block.timestamp + VOTING_PERIOD;

        emit ProposalCreated(proposalCount, _title);
        return proposalCount;
    }

    function calculateVotingWeight(address _voter) public view returns (uint256) {
        if (!voters[_voter].isRegistered) return 0;

        uint256 tokenBalance = governanceToken.balanceOf(_voter);
        uint256 reputation = voters[_voter].reputationScore;

        // Weighted formula: 40% tokens + 60% reputation
        return (tokenBalance * 40 / 100) + (reputation * 60 / 100);
    }

    function vote(uint256 _proposalId, bool _support)
        external nonReentrant {
        Proposal storage proposal = proposals[_proposalId];
        require(voters[msg.sender].isRegistered, "Not registered voter");
        require(block.timestamp <= proposal.endTime, "Voting period ended");
        require(!proposal.hasVoted[msg.sender], "Already voted");

        uint256 weight = calculateVotingWeight(msg.sender);
        require(weight > 0, "No voting power");

        proposal.hasVoted[msg.sender] = true;
```

```
        proposal.voteWeight[msg.sender] = weight;

        if (_support) {
            proposal.yesVotes += weight;
        } else {
            proposal.noVotes += weight;
        }

        emit VoteCast(_proposalId, msg.sender, _support, weight);
    }

    function getProposalResults(uint256 _proposalId)
        external view returns (
            string memory title,
            uint256 yesVotes,
            uint256 noVotes,
            bool isActive,
            bool passed
        ) {
        Proposal storage proposal = proposals[_proposalId];
        return (
            proposal.title,
            proposal.yesVotes,
            proposal.noVotes,
            block.timestamp <= proposal.endTime,
            proposal.yesVotes > proposal.noVotes
        );
    }
}
```

2.3 Governance Token Contract

solidity

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract GovernanceToken is ERC20, Ownable {
    constructor(string memory name, string memory symbol)
        ERC20(name, symbol) {}

    function mint(address to, uint256 amount) external onlyOwner {
        _mint(to, amount);
    }
}
```

Phase 3: Testing Strategy (Weeks 4-7)

3.1 Comprehensive Test Suite

javascript

```
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("Enhanced DAO Voting System", function () {
  let dao, token, owner, voter1, voter2, voter3;

  beforeEach(async function () {
    [owner, voter1, voter2, voter3] = await ethers.getSigners();

    // Deploy governance token
    const TokenFactory = await ethers.getContractFactory("GovernanceToken");
    token = await TokenFactory.deploy("DAOToken", "DAO");

    // Deploy DAO contract
    const DAOFactory = await ethers.getContractFactory("EnhancedDAOVoting");
    dao = await DAOFactory.deploy(await token.getAddress());

    // Setup initial state
    await token.mint(voter1.address, ethers.parseEther("100"));
    await token.mint(voter2.address, ethers.parseEther("50"));
    await token.mint(voter3.address, ethers.parseEther("25"));

    await dao.registerVoter(voter1.address, 80);
    await dao.registerVoter(voter2.address, 60);
    await dao.registerVoter(voter3.address, 40);
  });

  describe("Voting Weight Calculation", function () {
    it("Should calculate correct voting weight", async function () {
      const weight = await dao.calculateVotingWeight(voter1.address);
      // Expected: (100 * 0.4) + (80 * 0.6) = 40 + 48 = 88
      expect(weight).to.equal(88);
    });
  });

  describe("Proposal Lifecycle", function () {
    it("Should create and vote on proposals", async function () {
      await dao.createProposal("Test Proposal", "Testing enhanced voting");

      await dao.connect(voter1).vote(1, true);
      await dao.connect(voter2).vote(1, false);

      const results = await dao.getProposalResults(1);
```

```
        expect(results.yesVotes).to.be.gt(results.noVotes);
    });

    it("Should prevent double voting", async function () {
        await dao.createProposal("Double Vote Test", "Testing...");
        await dao.connect(voter1).vote(1, true);

        await expect(
            dao.connect(voter1).vote(1, false)
        ).to.be.revertedWith("Already voted");
    });
  });
});
```

Phase 4: ZKP Integration Planning (Weeks 8-12)

4.1 ZoKrates Setup

You'll need to install ZoKrates for zero-knowledge proofs:

```bash
# Install ZoKrates
curl -LSfs get.zokrat.es | sh

# Or using Docker
docker run -it zokrates/zokrates:latest
```

4.2 Privacy Circuit Design

Create a basic voting circuit (`vote_circuit.zok`):

```zok
```

```
import "hashes/sha256/512bitPacked" as sha256packed;

def main(private field vote, private field voterID, field nullifierHash) -> field {
    // Prove you know the vote without revealing it
    field[2] voteBits = [vote, voterID];
    field hash = sha256packed(voteBits)[0];

    // Verify nullifier to prevent double voting
    assert(nullifierHash == hash);

    return 1;
}
```

4.3 ZKP Integration Contract

solidity

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "./verifier.sol"; // Generated by ZoKrates

contract ZKDAOVoting is Verifier {
    mapping(uint256 => bool) public nullifiers;
    mapping(uint256 => uint256) public proposalVotes;

    function submitZKVote(
        uint[2] memory a,
        uint[2] memory a_p,
        uint[2][2] memory b,
        uint[2] memory b_p,
        uint[2] memory c,
        uint[2] memory c_p,
        uint[2] memory h,
        uint[2] memory k,
        uint[] memory inputs
    ) public {
        // Verify ZK proof
        require(verifyTx(a, a_p, b, b_p, c, c_p, h, k, inputs), "Invalid proof");

        // Extract nullifier from public inputs
        uint256 nullifier = inputs[0];
        require(!nullifiers[nullifier], "Vote already cast");

        // Record vote
        nullifiers[nullifier] = true;
        proposalVotes[1]++; // Simplified - actual implementation needs proposal ID
    }
}
```

Phase 5: DID Integration (Weeks 10-14)

5.1 DID Documentation Structure

javascript

```
// Example DID document structure
const didDocument = {
    "@context": ["https://www.w3.org/ns/did/v1"],
    "id": "did:example:123456789abcdefghi",
    "authentication": [{
        "id": "did:example:123456789abcdefghi#keys-1",
        "type": "Ed25519VerificationKey2020",
        "controller": "did:example:123456789abcdefghi",
        "publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
    }],
    "service": [{
        "id": "did:example:123456789abcdefghi#vcs",
        "type": "VerifiableCredentialService",
        "serviceEndpoint": "https://example.com/vc/"
    }]
};
```

5.2 Verifiable Credentials Integration

javascript

```
// VC verification in frontend
async function verifyCredential(vcJWT) {
    try {
        const credential = await didJWT.verifyJWT(vcJWT, {
            resolver: getResolver(),
            audience: 'did:example:dao'
        });

        return {
            valid: true,
            claims: credential.payload.vc.credentialSubject
        };
    } catch (error) {
        return { valid: false, error: error.message };
    }
}
```

Phase 6: Performance Testing & Gas Optimization (Weeks 12-16)

6.1 Gas Usage Testing

javascript

```
describe("Gas Usage Analysis", function () {
    it("Should measure gas consumption", async function () {
        const tx = await dao.createProposal("Gas Test", "Testing gas usage");
        const receipt = await tx.wait();
        console.log(`Proposal creation gas: ${receipt.gasUsed}`);

        const voteTx = await dao.connect(voter1).vote(1, true);
        const voteReceipt = await voteTx.wait();
        console.log(`Vote gas: ${voteReceipt.gasUsed}`);
    });
});
```

6.2 Performance Benchmarking

javascript

```
// Performance testing script
async function benchmarkSystem() {
    const startTime = Date.now();

    // Create multiple proposals
    for (let i = 0; i < 100; i++) {
        await dao.createProposal(`Proposal ${i}`, "Benchmark test");
    }

    const endTime = Date.now();
    console.log(`100 proposals created in ${endTime - startTime}ms`);
}
```

Phase 7: Frontend Development (Weeks 14-18)

7.1 React Setup

bash

```
npx create-react-app dao-frontend
cd dao-frontend
npm install ethers @rainbow-me/rainbowkit wagmi
```

7.2 Basic Frontend Structure

jsx

```
// App.js
import React from 'react';
import { WagmiConfig } from 'wagmi';
import { RainbowKitProvider } from '@rainbow-me/rainbowkit';
import VotingDashboard from './components/VotingDashboard';

function App() {
  return (
    <WagmiConfig config={wagmiConfig}>
      <RainbowKitProvider chains={chains}>
        <div className="App">
          <h1>Privacy-Preserving DAO</h1>
          <VotingDashboard />
        </div>
      </RainbowKitProvider>
    </WagmiConfig>
  );
}
```

Implementation Timeline & Milestones

Month 1: Foundation

- [x] Basic contract (you're here!)
- [ ] Enhanced voting mechanism
- [ ] Comprehensive testing
- [ ] Gas optimization

Month 2: Privacy Layer

- [ ] ZoKrates integration
- [ ] Privacy circuit development
- [ ] ZKP contract integration
- [ ] Privacy testing

Month 3: Identity Layer

- [ ] DID integration
- [ ] Verifiable credentials
- [ ] Sybil resistance testing
- [ ] Identity verification flow

Month 4: Integration & Evaluation

- [ ] Full system integration
- [ ] Performance benchmarking
- [ ] Security audit
- [ ] Documentation

Research Data Collection Strategy

Quantitative Metrics:

1. **Gas Consumption**: Measure for each operation
2. **Transaction Throughput**: Votes per second
3. **Proof Generation Time**: ZKP creation time
4. **Verification Time**: ZKP verification time
5. **Storage Costs**: On-chain data storage

Qualitative Analysis:

1. **Security Assessment**: Formal verification
2. **Usability Testing**: User interface evaluation
3. **Decentralization Analysis**: Power distribution metrics
4. **Privacy Analysis**: Information leakage assessment

Key Implementation Notes

1. **Start Simple**: Your current basic contract is perfect foundation
2. **Incremental Development**: Add one feature at a time
3. **Test Everything**: Write tests before adding complexity
4. **Document Progress**: Keep detailed research logs
5. **Version Control**: Use Git for all code changes

Recommended Learning Resources

1. **Solidity**: Cryptozombies, Solidity by Example
2. **ZKPs**: ZoKrates tutorial, ZK whiteboard sessions
3. **DIDs**: W3C DID specification, DIF resources
4. **Testing**: Hardhat documentation, Chai assertion library

Next Immediate Steps

1. Fix your current contract's double-voting issue
2. Add the governance token integration
3. Implement weighted voting mechanism
4. Write comprehensive tests
5. Begin ZoKrates experimentation

Remember: This is a significant research project. Focus on building incrementally and documenting your findings thoroughly. Each phase should produce measurable results for your thesis.

Good luck with your research! Feel free to ask specific questions about any phase.