**1. Introduction**

This document provides a comprehensive overview of the **Proof of Space (PoSpace)** smart contract prototype. The project was inspired by SpaceMint, aiming to create a sustainable alternative to Proof of Work (PoW) through disk space-based consensus mechanisms.

The contract was implemented in **Solidity**, tested using **Ganache**, and interacted with via **Remix IDE** and **MetaMask**. This documentation details the design, functionality, deployment, testing, and observations of the contract.

**2. Smart Contract Overview**

**2.1 Purpose**

The Proof of Space smart contract allows miners to:

* Commit their disk space using cryptographic proofs.
* Validate their proofs to earn rewards.
* Retrieve their reward balance proportional to the space committed.

This system reduces energy consumption by relying on disk space as the primary resource.

**2.2 Key Features**

* **Storage Commitments**: Miners submit a cryptographic hash and disk space size.
* **Proof Validation**: Proofs are verified to ensure they match the original commitment.
* **Reward Distribution**: Rewards are allocated proportionally to the committed disk space.

**3. Code Details**

**3.1 Contract Structure**

The contract includes the following key components:

**Data Structures**

solidity

struct Miner {

bytes32 commitment; // Cryptographic hash of the storage proof

uint256 spaceSize; // Size of committed disk space in GB

bool hasProven; // Indicates if the miner has successfully validated their proof

}

mapping(address => Miner) public miners; // Tracks commitments of all miners

mapping(address => uint256) public rewards; // Tracks rewards for each miner

uint256 public totalRewards = 1000 ether; // Total reward pool

**3.2 Core Functions**

**commitSpace**

solidity

function commitSpace(bytes32 \_commitment, uint256 \_spaceSize) external {

require(miners[msg.sender].commitment == bytes32(0), "Already committed");

require(\_spaceSize > 0, "Space size must be greater than zero");

miners[msg.sender] = Miner(\_commitment, \_spaceSize, false);

emit SpaceCommitted(msg.sender, \_spaceSize, \_commitment);

}

* **Purpose**: Allows miners to commit their storage proof and space size.
* **Validation**: Ensures miners cannot commit multiple times.
* **Event Emitted**: SpaceCommitted(address miner, uint256 spaceSize, bytes32 commitment).

**submitProof**

solidity

function submitProof(bytes32 \_proof) external {

require(miners[msg.sender].commitment != bytes32(0), "No commitment found");

require(!miners[msg.sender].hasProven, "Already proven");

bool isValid = \_proof == miners[msg.sender].commitment;

require(isValid, "Invalid proof");

miners[msg.sender].hasProven = true;

uint256 reward = miners[msg.sender].spaceSize \* 1 ether;

require(reward <= totalRewards, "Not enough rewards available");

rewards[msg.sender] += reward;

totalRewards -= reward;

emit ProofSubmitted(msg.sender, isValid, reward);

}

* **Purpose**: Validates miners’ proofs and distributes rewards.
* **Validation**: Compares \_proof with the stored commitment.
* **Event Emitted**: ProofSubmitted(address miner, bool valid, uint256 reward).

**getRewardBalance**

solidity

function getRewardBalance() external view returns (uint256) {

return rewards[msg.sender];

}

* **Purpose**: Allows miners to check their accumulated rewards.

**4. Deployment Details**

**4.1 Development Tools**

* **Language**: Solidity (v0.8.20)
* **Tools**: Remix IDE, Ganache, MetaMask
* **Testing Environment**: Ganache (local Ethereum blockchain)

**4.2 Deployment Steps**

1. Write and compile the contract in **Remix IDE**.
2. Deploy the contract using Remix, connected to **Ganache** via **MetaMask**.
3. Verify successful deployment by obtaining the contract address.

**5. Testing**

**5.1 Testing Setup**

* **Environment**: Local blockchain simulated with Ganache.
* **Test Accounts**: Pre-funded accounts from Ganache.

**5.2 Test Scenarios**

**Test Case 1: Commit Space**

* **Input**: Commitment hash (web3.utils.keccak256("test-storage-proof")), space size (10 GB).
* **Expected Result**: Commitment successfully stored, event SpaceCommitted emitted.

**Test Case 2: Submit Proof**

* **Input**: Proof matching the stored commitment.
* **Expected Result**: Proof validated, miner rewarded, event ProofSubmitted emitted.

**Test Case 3: Check Reward Balance**

* **Input**: Call getRewardBalance().
* **Expected Result**: Displays the correct reward balance in ETH.

**6. Security Considerations**

* **Fake Proof Prevention**: Proofs are validated against commitments to ensure authenticity.
* **Reward Cap**: Ensures rewards do not exceed the total available pool.
* **Duplicate Commit Prevention**: Each miner can commit only once.

**7. Observations and Future Work**

**7.1 Observations**

* The prototype functions correctly for small-scale testing on Ganache.
* The reward system and proof validation are reliable and efficient.

**7.2 Future Work**

* **Scalability**: Deploy and test on public Ethereum testnets (e.g., Sepolia, Goerli).
* **Advanced Proof Mechanisms**: Implement Merkle trees for more secure proof validation.
* **Real-World Integration**: Combine with decentralized storage solutions like IPFS.

**8. Conclusion**

This project demonstrates the feasibility of Proof of Space as an energy-efficient blockchain consensus mechanism. By leveraging disk space, PoSpace reduces the environmental impact of mining while maintaining decentralization and security. The prototype lays the groundwork for further exploration and real-world applications.