Decentralized Reputation System

Introduction

The project design consists of a Staking Protocol that requires implementing a Reputation System in the EVM smart contract. The smart contract will be developed using Solidity and deployed on the Ethereum blockchain. Measures will be taken to handle scalability, security, and best practices from the decentralized finance.

Overview

Decentralized Systems have the potential to deliver a transparent, secure, and robust system that can handle financial transactions diligently. However, there are challenges like *scalability limitations*, *high transaction fees,* and *security concerns*, which come along with this implementation and need proper attention. It is recommended to implement zk Proofs and make use of Oracle in the decentralized system, such that we can make the best use of the Reputation System.

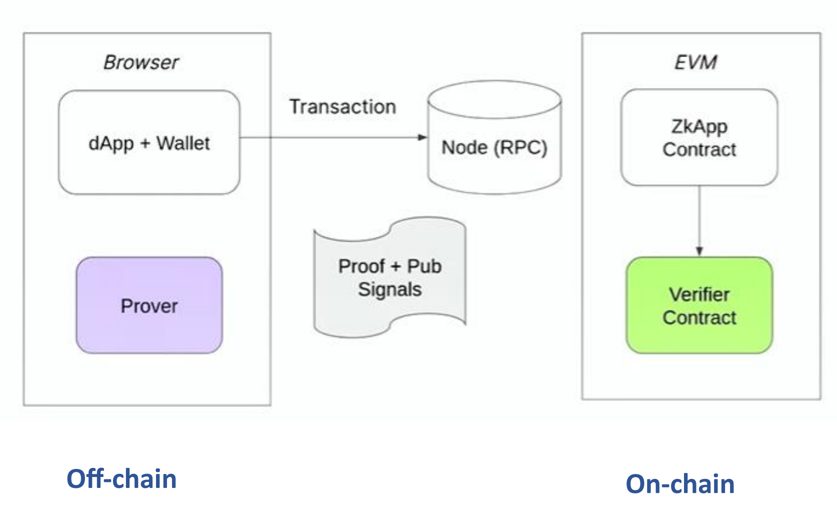
Technical Considerations:

zk Proofs

Introducing zero-knowledge proofs (ZKP) into the blockchain architecture can help enhance the scalability and security of the smart contract. ZKP are cryptographic protocols that help with the secrecy and privacy of the data.

ZKP has some on-chain and off-chain components. This helps with network scalability as there is less load on the network. This also helps in keeping the data secure since the attacker will attack on-chain and not have access to the off-chain computations that are encrypted.

Basic zk-Proof Design



Off-chain transactions do not require network costs and transactions are not affected by consensus delays and blockchain transaction queues.

The prover and dApp are at the client end. The `public signals` and `proof` are sent to Node JS as `Transaction` to EVM. The verifier utilizes the public signals and proof along with the ZK app contract to verify the proofs.

zk SNARK

zk-SNARK is the most efficient form of ZKP which could be used in this architecture. zk-SNARK stands for '*zero-knowledge Succinct Non-Interactive Arguments of Knowledge*'.

The decentralized financial system can have several key components like hierarchical smart contracts, off-chain payment networks, and partitioned storage nodes, that zk-SNARK can help with.

Some imminent contributions that can be made by zk-SNARK in this project:

a. The mathematical formula used for the calculation of the payment cycle could make use of zk-SNARK which ensures the validity of transactions.

b. Many decentralized finance projects consist of hierarchical smart contracts. Zk-SNARK and hierarchical smart contracts can function together to help verify financial methods.

c. Off-chain payment methods and transactions help with network scalability solutions as it reduces the load on the network and certain parts of the data are encrypted.

d. Wallet State Proof and transaction proof of each user can be stored in the storage nodes off-chain.

e. If there is a dispute, the proofs submitted by zk-SNARK can be used for the resolution of the disputes between different parties.

Reputation System

Understanding Reputation System

Reputation Systems are designed to quantify and assess the reliability, quality, and trustworthiness of users or services in a network. In the reputation system, trusteeship data is the most important data, there should be a mechanism to monitor the participant's credibility and reputation.

A staking pool thrives on different predictions and based on these predictions the tokens are bought or sold in a liquidity pool. Prediction of a seller generating profit in a pool or predicting the stakers will stake in the pool could increase a user's position in a pool.

There could be stakers just making noise in the pool. The reputation system could reduce their position in the liquidity pool. This motivates the stakers to focus on the pool and generate positive and accurate responses.

Reputation Score

The reputation score of a staker shows that the person is a valuable member of the pool and not an attacker or noise maker. People may wish to follow such a person's view and it would give them direction and spread positivity in the pool.

The reputation score could be programmatically calculated based on certain parameters. The parameters could be the person's behavior in this pool as well as outside the pool. A person with a bad reputation outside the pool is least expected a good behavior in the pool. Data of the person outside the pool can be checked using Oracle. There should be care to prevent any Sybil attack.

Scalability Challenges

The reputation of each person should be recognized by DAO. The reputation score can be calculated off-chain to reduce the load on the blockchain network. However, this would question the authenticity of the data and this can be handled by zk-SNARK, which we discussed earlier. The offloading of computations to off-chain handles the scalability challenges with the reputation system. Some decentralized finance systems provide reputation tokens to the users for good behavior.

External Interaction

Reputation System could use data from outside to validate the users and these data could be obtained from Oracle. It could be some users are staking in different defi protocol platforms or are skilled in decentralized trading, and have a good reputation. In that case, we could use the Oracle data. However, an external data again questions the trustworthiness of the data. It can be achieved by - *Cryptographic Signatures* and *Consensus Mechanism*. Cryptographic signature verifies the authenticity of the data and consensus mechanism validates the new entries and preserves their immutability.

Security Measures

Sybil attack occurs when multiple fake identities could compromise a protocol. This attack could completely change the actual working mechanism of the protocol. As the protocol gets older and more popular, the attackers would want to come forward to take charge of the protocol.

We benefit from the reputation system here as the reputation scores given to the older members assists in validating honest members and the system has the power to overthrow Sybil nodes. Security measures can be handled by proper implementation of a reputation system.

User Experience Integration

We have reputation systems in the web2 space and it helps us interact with the entities in a better way. The entities could be an app, a web page, or an online post. One of the first reputation score is the page rank used by Google in web2 space to help give its users the best web page out of the millions of pages online. Hence, similarly in web3 space we require the reputation score or ranking to help us gain confidence with a certain protocol or dApp. Users will have greater trust in the system has an integrated reputation mechanism.

Tools/Utilities

Tools required to build this Staking Protocol are – VScode IDE, Oracle, Decentralized Storage (IPFS, Arweave).

Languages – Solidity will be used to code the program and Foundry will be used to test the program. Other static testing tools such as Slither and Aderyn will be used for checking bugs in the program.

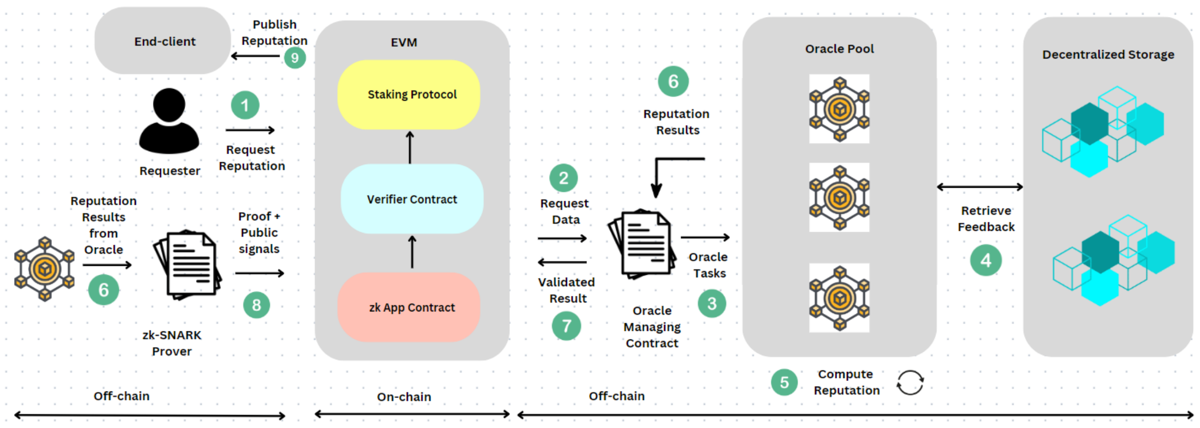
Libraries – OpenZeppelin and Chainlink. Any other library required will be added as per the requirement.

Environment – The project will be deployed on Ethereum.

Staking protocol Technical Architecture

(Reputation System, zk-SNARK, Oracle, Decentralized Storage)

The requester requests a reputation score to the Staking Protocol contract, which requests data from the Oracle system. The Oracle system retrieves feedback from decentralized storage and computes the reputation score. This reputation score is further sent to the zk-SNARK prover contract, which adds the proof with public signals and sends it to the zk App contract. The data is sent to the verifier contract of zk-SNARK (that is on-chain). It further computes and publishes the result to the end client.



We can observe the process is divided into on-chain and off-chain. The off-chain computations help to take off the load from the blockchain system and prevent consensus delays.

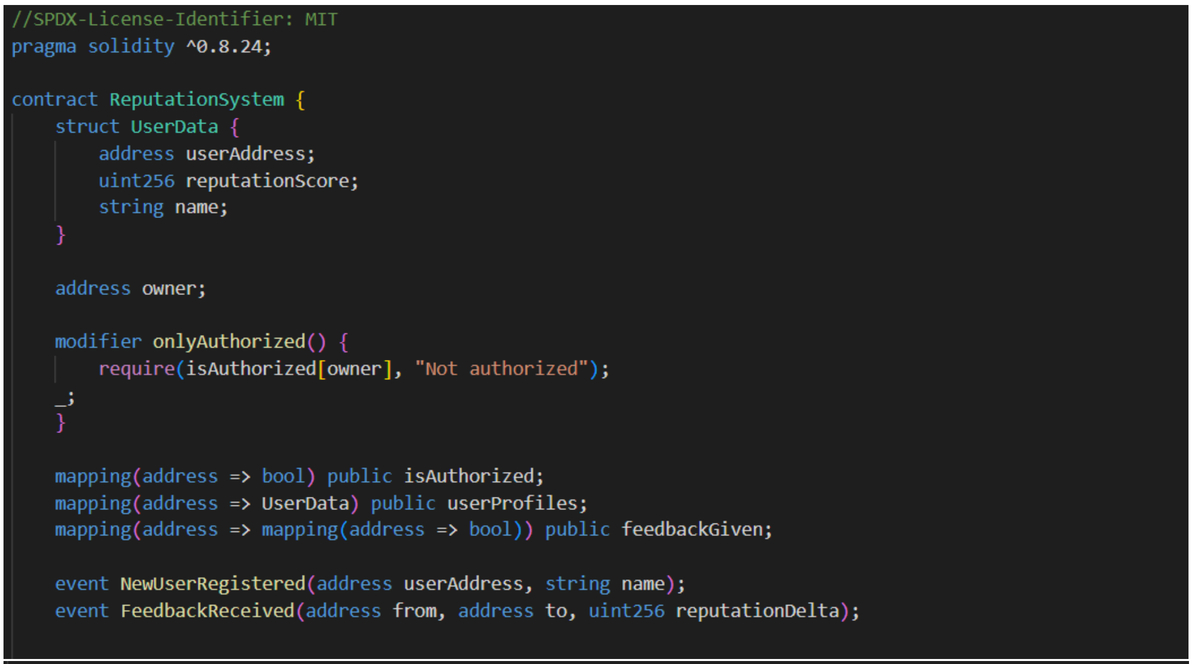
Code Snippet

Below is the code snippet for a basic Reputation System. All the users can give feedback to the other users in the organization and based on that the reputation score of a particular user is enhanced. Users can choose not to give feedback to a particular user, it is not mandatory.

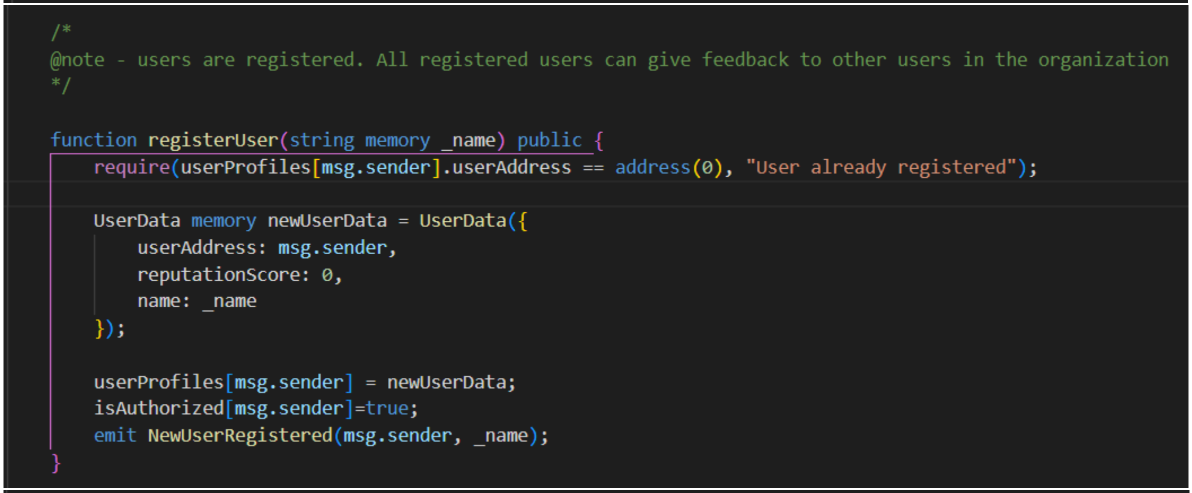
The program checks for the authorization of the user before giving feedback. The user must be registered to provide feedback to other users.

There can be feedback from numerous sources such as external interactions like Oracles and that feedback can be incorporated into user feedback in real-time scenarios. More parameters could be added to validate the user and these could be added to the user data profile.

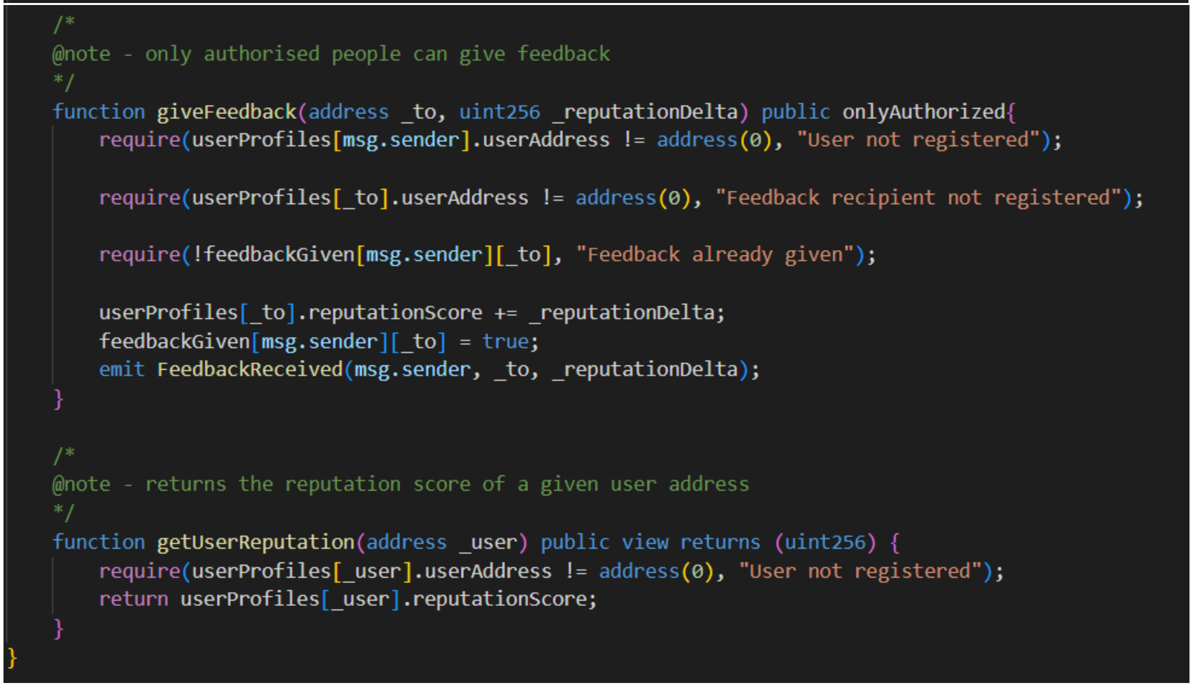
There could be reputation tokens awarded to the users instead of just reputation scores in real time. The timestamp when the user joined the organization and if the user has a better reputation in other organizations as well, could be taken into consideration.



```solidity
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

contract ReputationSystem {
    struct UserData {
        address userAddress;
        uint256 reputationScore;
        string name;
    }

    address owner;

    modifier onlyAuthorized() {
        require(isAuthorized[owner], "Not authorized");
    _;
    }

    mapping(address => bool) public isAuthorized;
    mapping(address => UserData) public userProfiles;
    mapping(address => mapping(address => bool)) public feedbackGiven;

    event NewUserRegistered(address userAddress, string name);
    event FeedbackReceived(address from, address to, uint256 reputationDelta);
```



```solidity
    /*
    @note - users are registered. All registered users can give feedback to other users in the organization
    */

    function registerUser(string memory _name) public {
        require(userProfiles[msg.sender].userAddress == address(0), "User already registered");

        UserData memory newUserData = UserData({
            userAddress: msg.sender,
            reputationScore: 0,
            name: _name
        });

        userProfiles[msg.sender] = newUserData;
        isAuthorized[msg.sender]=true;
        emit NewUserRegistered(msg.sender, _name);
    }
```



```solidity
    /*
    @note - only authorised people can give feedback
    */
    function giveFeedback(address _to, uint256 _reputationDelta) public onlyAuthorized{
        require(userProfiles[msg.sender].userAddress != address(0), "User not registered");

        require(userProfiles[_to].userAddress != address(0), "Feedback recipient not registered");

        require(!feedbackGiven[msg.sender][_to], "Feedback already given");

        userProfiles[_to].reputationScore += _reputationDelta;
        feedbackGiven[msg.sender][_to] = true;
        emit FeedbackReceived(msg.sender, _to, _reputationDelta);
    }

    /*
    @note - returns the reputation score of a given user address
    */
    function getUserReputation(address _user) public view returns (uint256) {
        require(userProfiles[_user].userAddress != address(0), "User not registered");
        return userProfiles[_user].reputationScore;
    }
}
```