Plastik Token Governance and Voting Smart Contract

Overview

1. **Governance Tokens:** When the contract starts, it integrates with the PLASTIK token, an ERC-20 token. Users who hold at least 10,000 tokens in their wallet can participate in governance.
2. **Proposing Changes:** Only token holders can propose changes to a "law" (some on-chain rule or policy).
3. **Voting Process:** Eligible token holders can vote "yes" or "no" on a proposal.
4. **Decision Making:** After the voting deadline, the contract counts the votes and either accepts or rejects the proposal.
5. **State Management:** The contract maintains a state that records the current law, the voting process, and associated constraints.

Key Components

1. **GovState:** Represents the contract's state, storing:
    - `law`: The current rule or policy.
    - `mph`: A minting policy hash for voting tokens.
    - `voting`: Holds the current active proposal and votes (if any).
2. **Proposal:** Defines a suggested law change with:
    - `newLaw`: The proposed law text.
    - `tokenName`: A voting token identifier.
    - `votingDeadline`: Time until which voting is allowed.
3. **Voting:** Tracks votes cast by token holders.
4. **GovInput (User Actions):**
    - `MintTokens [TokenName]`: Mints voting tokens.
    - `ProposeChange Address Proposal`: Submits a proposal.
    - `AddVote Address TokenName Bool`: Allows voting.
    - `FinishVoting`: Finalizes the voting process.
    - `Check`: (Likely for debugging or state checking).

How the Contract Works

1. **Initialization:**
    - The contract interacts with the PLASTIK token, an ERC-20 token.
    - Users with at least 10,000 tokens in their wallet are eligible to vote.
2. **Proposing a Law Change:**
    - A token holder submits a new proposal.
    - The contract verifies that the proposer owns at least 10,000 PLASTIK tokens.
    - If valid, the proposal is stored in the contract state.

3. **Voting Process:**
    - Token holders vote by submitting true (yes) or false (no).
    - Each token holder can vote once per proposal.
4. **Finalizing the Vote:**
    - When the deadline is reached, the contract counts "yes" votes.
    - If they meet or exceed requiredVotes, the new law is adopted.
    - Otherwise, the proposal is discarded.

Key Smart Contract Functions

- **transition:** Defines how the state changes in response to different inputs.
- **ownsVotingToken:** Ensures only token holders with at least 10,000 PLASTIK tokens can vote.
- **mkTokenName:** Generates token names dynamically.
- **typedValidator:** Creates a validator script for secure state transitions.
- **contract:** Main contract logic handling new laws and voting.

Plastik Token Escrow Smart Contract

This escrow smart contract in Plutus is designed to facilitate secure value exchanges between multiple parties, ensuring that funds are only released under specified conditions. Here's a breakdown of its key functionalities:

1. Contract Overview

- The escrow contract locks funds from contributors until specific conditions are met.
- Funds can be redeemed (released to predefined targets) if the conditions are satisfied.
- If the conditions are not met by the deadline, contributors can get a refund.

2. Key Components

Data Structures

- **`EscrowTarget`**: Specifies where funds should go, either to a public key address or a script address.
- **`EscrowParams`**: Defines the escrow terms, including:
    - `escrowDeadline`: The latest time funds can be spent.
    - `escrowTargets`: The list of recipients and their required amounts.
- **`Action`**: Defines two possible actions:
    - `Redeem`: Release funds to predefined targets.
    - `Refund`: Allow contributors to reclaim their funds after the deadline.

Smart Contract Logic

- **`validate` Function**:

    - If `Redeem` is called:
        - Ensures the transaction occurs after the `escrowDeadline`.
        - Confirms that all targets receive their specified funds.
    - If `Refund` is called:
        - Ensures the transaction occurs before the `escrowDeadline - 1`.
        - Confirms that the contributor (who deposited funds) is reclaiming them.
- **Transaction Constraints**:
    - Uses `mustPayToPubKey` for direct payments.
    - Uses `mustPayToOtherScriptWithDatumInTx` for script-based payments.

3. Workflow

1. **Paying into the Escrow**:
    - Users send funds to the escrow contract.
2. **Redeeming Funds**:
    - If the conditions are met (correct amount & valid deadline), the funds are sent to the recipients.
3. **Refunding Funds**:
    - If the deadline passes and conditions are not met, contributors can reclaim their funds.

4. Security Measures

- Uses `traceIfFalse` for debugging.
- Ensures only authorized parties can redeem/refund.
- Prevents unauthorized access by validating signatures.