ANASTASIA LABS

Proof of Achievement - Milestone 1

Upgradable Multi-Signature Smart Contract

Project Number: 1100025
Project Manager: Jonathan Rodriguez
Project Name: Anastasia Labs X Maestro - Plug ‘n Play 2.0
URL: Catalyst Proposal

Contents

1. Introduction

Our project aims to develop an upgradable multi-signature contract for secure transactions on the Cardano blockchain. This contract requires multiple signatures to authorize transactions, with adjustable thresholds based on signatory requirements.

Key features include:

- Setting up a multi-signature wallet with authorized signatories
- Defining and adjusting the threshold for required signatures
- Updating the list of signatories
- Enforcing spending limits for transactions

This report demonstrates our progress in implementing Secure Spending of Assets, Seamless Adjustment of Signer Thresholds, and Dynamic Addition or Removal of Signers.

2. Secure Spending of Assets

Our upgradable multi-signature smart contract has been rigorously tested to ensure that asset transactions are executed only by authorized members. The contract enforces a multi-signature requirement, necessitating approval from a predefined number of signatories before any transaction can be executed. Our comprehensive testing suite has validated the contract's ability to manage secure asset spending effectively

2.1. Detailed Test Analysis

2.1.1. Test Case: Successful Transaction

We've implemented a test case to validate the contract's ability to execute transactions when properly authorized and the implementation can be viewed on the test success_sign code

```bash
Testing ...

    ┌─ multisig ──────────────────────────────────
    │ PASS [mem: 589296, cpu: 274524061] success_sign
    │ · with traces
    │ | Test: Successful Transaction Signing
    │ | Total number of signatories in the multisig
    │ | 4
    │ | Required threshold of signatures
    │ | 3
    │ | Number of signatories actually signing this transaction
    │ | 3
    │ | Total amount in the contract (in lovelace)
    │ | 100000000000
    │ | Maximum spending limit per transaction (in lovelace)
    │ | 1000000000
    │ | Amount being withdrawn in this transaction (in lovelace)
    │ | 1000000000
    │ | Remaining amount in the contract after withdrawal (in lovelace)
    │ | 99000000000
    │ | Result: Transaction successfully signed and executed!
    └──────────────────────────────────── 1 tests | 1 passed | 0 failed
```

This test validates the contract's ability to execute transactions when properly authorized. It demonstrates a successful transaction where:

- 3 out of 4 signatories approved the transaction (meeting the threshold).
- The withdrawal amount was within the spending limit.
- The contract balance was correctly updated after the transaction.

The test confirms that the contract correctly processes transactions when the required number of signatories (3 out of 4) approve and the withdrawal amount is within the specified limit.

2.1.2. Test Case: Rejected Insufficient Signatures

To demonstrate the contract's security measures, we've implemented a test case for rejecting transactions with insufficient signatures with the implementation on the test reject_insufficient_signatures code

```bash
Testing ...

┌─ multisig ──────────────────────────────────────────────────────
│ PASS [mem: 410732, cpu: 207334934] reject_insufficient_signatures
│ · with traces
│ | Test: Rejecting Transaction with Insufficient Signatures
│ | Total number of signatories in the multisig
│ | 4
│ | Required threshold of signatures
│ | 3
│ | Number of signatories actually signing this transaction
│ | 2
│ | Total amount in the contract (in lovelace)
│ | 100000000000
│ | Maximum spending limit per transaction (in lovelace)
│ | 1000000000
│ | Attempted withdrawal amount (in lovelace)
│ | 1000000000
│ | Contract amount if withdrawal were allowed (in lovelace)
│ | 99000000000
│ | Result: Transaction rejected due to insufficient signatures!
│ | signed_within_threshold ? False
└──────────────────────────────────────── 1 tests | 1 passed | 0 failed
```

This test demonstrates the contract's robust security measures by successfully rejecting a transaction with insufficient signatures (2 provided, 3 required), thereby protecting assets from unauthorized spending.

3. Seamless Adjustment of Signer Thresholds

The smart contract features a robust and user-friendly mechanism that allows authorized members to adjust the signer thresholds without compromising security. This functionality is crucial for adapting to evolving security requirements and organizational changes. The process ensures that any adjustments to the number of required signatures for transaction approval are carried out smoothly and securely. Authorized members can modify the threshold with ease, while the contract's built-in security measures maintain the integrity of the approval process. This adaptability is vital for maintaining both the flexibility and security of the multi-signature system as conditions and needs change.

3.1. Threshold Adjustment Workflow

1. **Initiate Threshold Change:** Users can propose new threshold values through an intuitive interface.
2. **Review Proposed Changes:** The system clearly presents current and proposed thresholds for easy comparison.
3. **Collect Required Signatures:** Authorized signers can efficiently review and sign the proposal.
4. **Confirmation of Update:** Upon collecting the required signatures, the system promptly updates the threshold.

This streamlined process ensures that threshold adjustments are both secure and user-friendly.

3.2. Detailed Test Analysis

3.2.1. Test Case: Successful Threshold Adjustment

We've implemented a test case to verify the contract's ability to adjust the signer threshold the following link: test success_adjust_threshold code

```bash
Testing ...

┌─ multisig ──────────────────────────────────────────────
│ PASS [mem: 640878, cpu: 267881842] success_adjust_threshold
│ · with traces
│ | Test: Successfully Adjusting Signature Threshold
│ | Original signature threshold
│ | 2
│ | Total number of signatories in the multisig
│ | 4
│ | Current contract value (in lovelace)
│ | {_ h'': {_ h'': 100000000000 } }
│ | User clicks on action Redeemer: Update
│ | 122([])
│ | New threshold value
│ | 3
│ | Number of signatories approving this change
│ | 4
│ | Contract value after threshold adjustment (should be unchanged)
│ | {_ h'': {_ h'': 100000000000 } }
│ | Result: Signature threshold successfully updated!
└──────────────────────────────────────── 1 tests | 1 passed | 0 failed

  Summary 1 check, 0 errors, 0 warnings
```

This test verifies the contract's ability to adjust the signer threshold from 2 to 3 without errors, demonstrating the flexibility of the contract's security parameters.

4. Dynamic Addition and Removal of Signers

Showcase of the smart contract's capability to dynamically add or remove signers as needed. The contract allows for flexible management of the signer pool, adapting to organizational changes while maintaining security.

4.1. Dynamic Signer Addition Process

1. **Proposal Initiation:** An authorized user proposes a new signer and provides the necessary credentials.
2. **Collective Review:** Existing signers review the proposal details.
3. **Approval Gathering:** The system collects the required signatures for the addition.
4. **Execution and Verification:** Upon approval, the new signer is added, and the updated list is displayed for confirmation.

4.1.1. Detailed Test Analysis

4.1.1.1. Test Case: Successful Signer Addition

We've implemented a test case to confirm the contract's ability to add a new signer on the test success_add_signer code

```bash
Testing ...

┌─ multisig ──────────────────────────────────────
│ PASS [mem: 596818, cpu: 258625029] success_add_signer
│ · with traces
│ | Test: Successfully Adding a New Signer to the Multisig Contract
│ | Number of signatories before addition
│ | 4
│ | Current contract value (in lovelace)
│ | 100000000000
│ | Current signature threshold
│ | 3
│ | Number of signatories approving this change
│ | 4
│ | Redeemer used for this operation
│ | 122([])
│ | Number of signatories after addition
│ | 5
│ | Signature threshold after addition (unchanged)
│ | 3
│ | Contract value after adding signer (should be unchanged)
│ | 100000000000
│ | Result: New signer successfully added to the multisig!
└──────────────────────────────────── 1 tests | 1 passed | 0 failed
```

This test confirms the contract's ability to dynamically add a new signer, increasing the total number of signatories from 4 to 5.

4.2. Dynamic Signer Removal Process

The Multi-sig Contract is the primary contract responsible for managing the list of authorized signers, validating transactions, and ensuring the proper execution of multi-sig operations.

- **Removal Proposal:** An authorized user initiates the removal of a specific signer.
- **Proposal Review:** Remaining signers assess the removal proposal.
- **Consensus Building:** The system gathers necessary approvals, excluding the signer in question.
- **Execution and Confirmation:** Post-approval, the signer is removed, and the system displays the updated list for verification.

4.3. Detailed Test Analysis

4.3.1. Test Case Scenario: Successful Removal of Signer

The test case code can be found on the test success_remove_signer code

```bash
Testing ...

    ┌─ multisig ──────────────────────────────────────────
    │ PASS [mem: 525925, cpu: 233410760] success_remove_signer
    │ · with traces
    │ | Test: Successfully Removing a Signer from the Multisig Contract
    │ | Current contract value (in lovelace)
    │ | 100000000000
    │ | Number of signatories before removal
    │ | 4
    │ | Signature threshold before removal
    │ | 3
    │ | Number of signatories approving this change
    │ | 4
    │ | Redeemer used for this operation
    │ | 122([])
    │ | Number of signatories after removal
    │ | 3
    │ | Signature threshold after removal and adjustment (Can be changed)
    │ | 2
    │ | Contract value after removing signer (should be unchanged)
    │ | 100000000000
    │ | Result: Signer successfully removed and threshold adjusted
    └──────────────────────────────────────── 1 tests | 1 passed | 0 failed
```

This test verifies the contract's capability to remove signers and automatically adjust the threshold, demonstrating its adaptability to changing organizational needs while maintaining security.

4.4. Security Considerations

The contract maintains a minimum signer threshold to prevent security vulnerabilities. Threshold adjustments may be required before certain signer removals to maintain contract integrity.

5. Conclusion

Our upgradable multi-signature contract successfully meets all the acceptance criteria:

1. Secure Spending of Assets: We've demonstrated that only authorized members can execute transactions, with robust checks for signature thresholds and spending limits.
2. Seamless Adjustment of Signer Thresholds: Our contract allows for easy adjustment of signature thresholds while maintaining security, as shown in our threshold adjustment test.
3. Dynamic Addition or Removal of Signers: We've implemented and tested functionality for both adding and removing signers, showcasing the contract's flexibility in managing the signer pool.

All documentation, including detailed test cases and explanations, is available in our Aiken Upgradable Multisig Github Repo

This upgradable multi-signature contract provides a secure, flexible, and user-friendly solution for managing shared assets on the Cardano blockchain.