ANASTASIA LABS

# Upgradable Multi-Signature Contract

# Project Design Specification

# Contents

**1. Overview ..... 1**

**2. Architecture ..... 2**

**3. Specification ..... 3**
3.1. System Actors ..... 3
3.2. Tokens ..... 3
3.3. Smart Contracts ..... 4
3.3.1. Multisig Multi-validator ..... 4

**4. Transactions ..... 8**
4.1. Mint :: InitMultiSig ..... 8
4.1.1. Inputs ..... 8
4.1.2. Mints ..... 9
4.1.3. Outputs ..... 9
4.2. Mint :: EndMultiSig ..... 10
4.2.1. Inputs ..... 10
4.2.2. Mints ..... 11
4.2.3. Outputs ..... 11
4.3. Spend :: Sign ..... 12
4.3.1. Spend :: Update ..... 14

# 1. Overview

The Upgradable Multi-Signature Smart Contract is developed using Aiken and is designed to facilitate collaborative management of funds and operations on the Cardano blockchain. This contract allows multiple authorized signers to collectively approve and execute transactions, manage funds, and update the multi-sig configuration (such as adding or removing signers or changing the signature threshold).

The contract ensures secure and efficient transactions by through the validation of signatories, enforcing spending limits, and allowing for updates to the signer list and thresholds within a decentralized framework.

## 2. Architecture

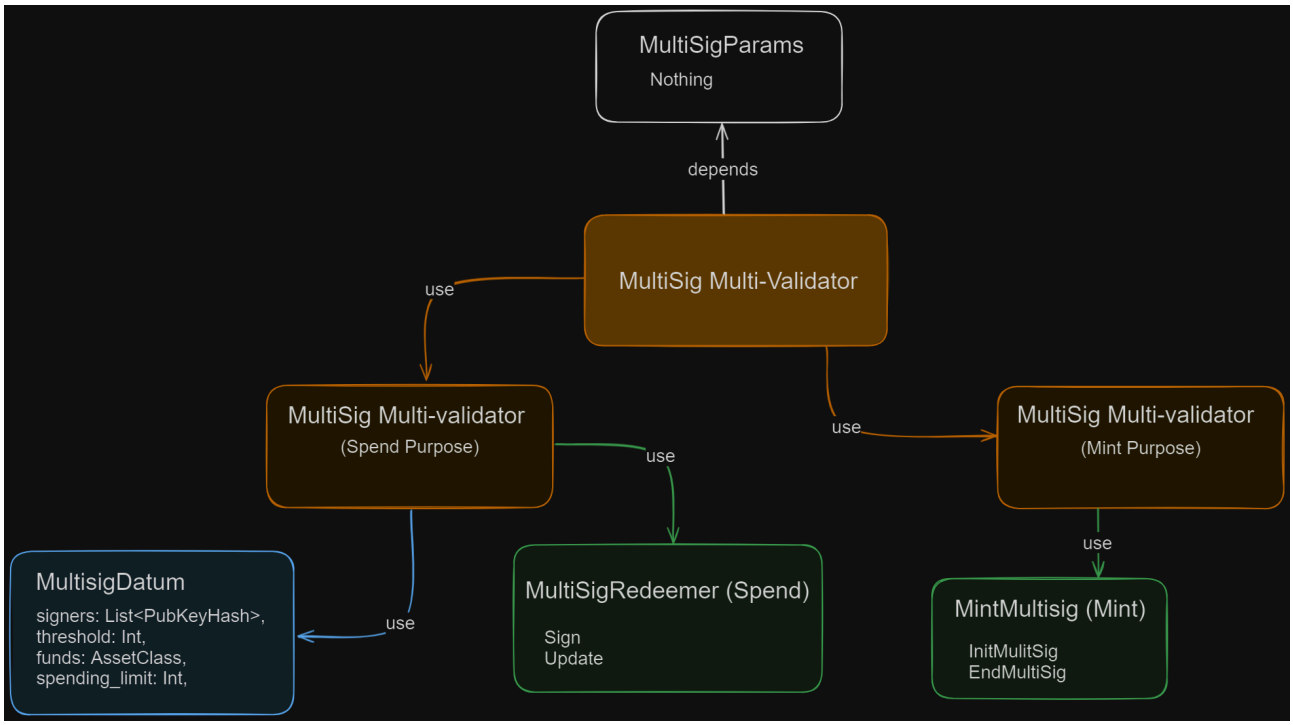



Figure 1: Upgradable Multisig Architecture

The architecture consists of a single multi-validator contracts that manages the mulit-signature functionality in this system.

1. **Multisig Contract**

   This is the core contract responsible for handling the logic for:

   - Managing authorized signers
   - Validating and executing transactions
   - Handling contract upgrades
   - Verifying signatures and authenticating transactions

# 3. Specification

## 3.1. System Actors

- **Initiator**

  The entity who creates the initial multi-sig wallet setup, defining the initial set of signers and thresholds. The initiator initiates the multi-sig with the datum and mints an Multisig NFT.

- **Signers**

  Entities who are authorized to sign transactions, participate in the management of the multi-sig wallet, and approve changes to the signer list or thresholds. Authorization is based on their inclusion in the signers list within the contract's datum.

## 3.2. Tokens

- **Multisig NFT**

  A unique Non-Fungible Token (NFT) representing the state and authority of the multi-sig contract. The Multisig NFT ensures the uniqueness and integrity of the contract's UTxO and is used to control access and operations within the contract.

  - **Token Name:** Derived from the transaction hash (TxHash) and output index (TxIx) of the UTXO where the NFT is initially minted.

## 3.3. Smart Contracts

### 3.3.1. Multisig Multi-validator

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 using the multisig NFT.

#### 3.3.1.1. Parameters

- None

#### 3.3.1.2. Mint Purpose

##### 3.3.1.2.1. Redeemer

```
pub type MintMultisig {
  InitMultiSig {
    output_reference: OutputReference,
    input_index: Int,
    output_index: Int,
  }
  EndMultiSig { datum: MultisigDatum, input_index: Int }
}
```

- **InitMultiSig:** Creates a new multi-sig setup by minting exactly one Multisig NFT.
- **EndMultiSig:** Terminates a multi-sig setup by burning exactly one Multisig NFT.

##### 3.3.1.2.2. Validation

1. **InitMultiSig**

   Allows the initiator to create a new multi-sig setup by minting exactly one Multisig NFT.

   - Validate that out_ref (output reference) must be present in the transaction inputs.

- Validate that the redeemer mints exactly one Multisig NFT with the correct token name derived from the UTXO.
- Ensure that the Multisig NFT is locked at the Multi-Sig Validator script address.
- Validate that the initial datum is correctly set with the signers, threshold, and other configuration parameters.

2. **EndMultiSig:**

   Allows the authorized signers to terminate the multi-sig contract by burning the Multisig NFT.

   - Verify that the required number of authorized signers have signed the transaction.
   - Ensure that the Multisig NFT is present in the input and is being burned in the transaction.
   - Confirm that all funds are appropriately distributed upon termination.

#### 3.3.1.3. Spend Purpose

The contract uses the Spend purpose, allowing it to handle spending operations such as signing transactions and updating the multi-sig configuration.

#### 3.3.1.4. Datum

The multisig datum structure holds the current state of the multisig arrangement:

```
pub type MultisigDatum {
  signers: List<PubKeyHash>,
  threshold: Int,
  funds: AssetClass,
  spending_limit: Int,
}
```

- **`signers`:** List of public key hashes of authorized signers.
- **`threshold`:** Minimum number of required signatures.
- **`funds`:** AssetClass of the funds to be withdrawn.
- **`spending_limit`:** Max Amount of funds to be withdrawn per transaction.

#### 3.3.1.5. Redeemer

The contract supports two types of operations, represented by the redeemer:

```
pub type MultisigRedeemer {
  Sign { input_index: Int, output_index: Int }
  Update { input_index: Int, output_index: Int }
}
```

- **Sign:** For executing fund transfers
- **Update:** For modifying the multisig configuration

#### 3.3.1.6. Validation

1. **Sign**

   The redeemer allows a majority of the authorized signers to collectively approve and execute transactions using the funds controlled by the multi-signature contract

   - Verifies that the required number of authorized signers have signed the transaction (based on the threshold).
   - Ensures the transfer amount does not exceed the `spending_limit`
   - Checks that the Multisig NFT remains at the script address in the output.
   - Checks that the total value is preserved across inputs and outputs (excluding the spent amount).
   - Ensure the output datum matches the input datum (no changes to the multisig configuration)

2. **Update**

   The redeemer enables the modification of the multi-signature arrangement itself.

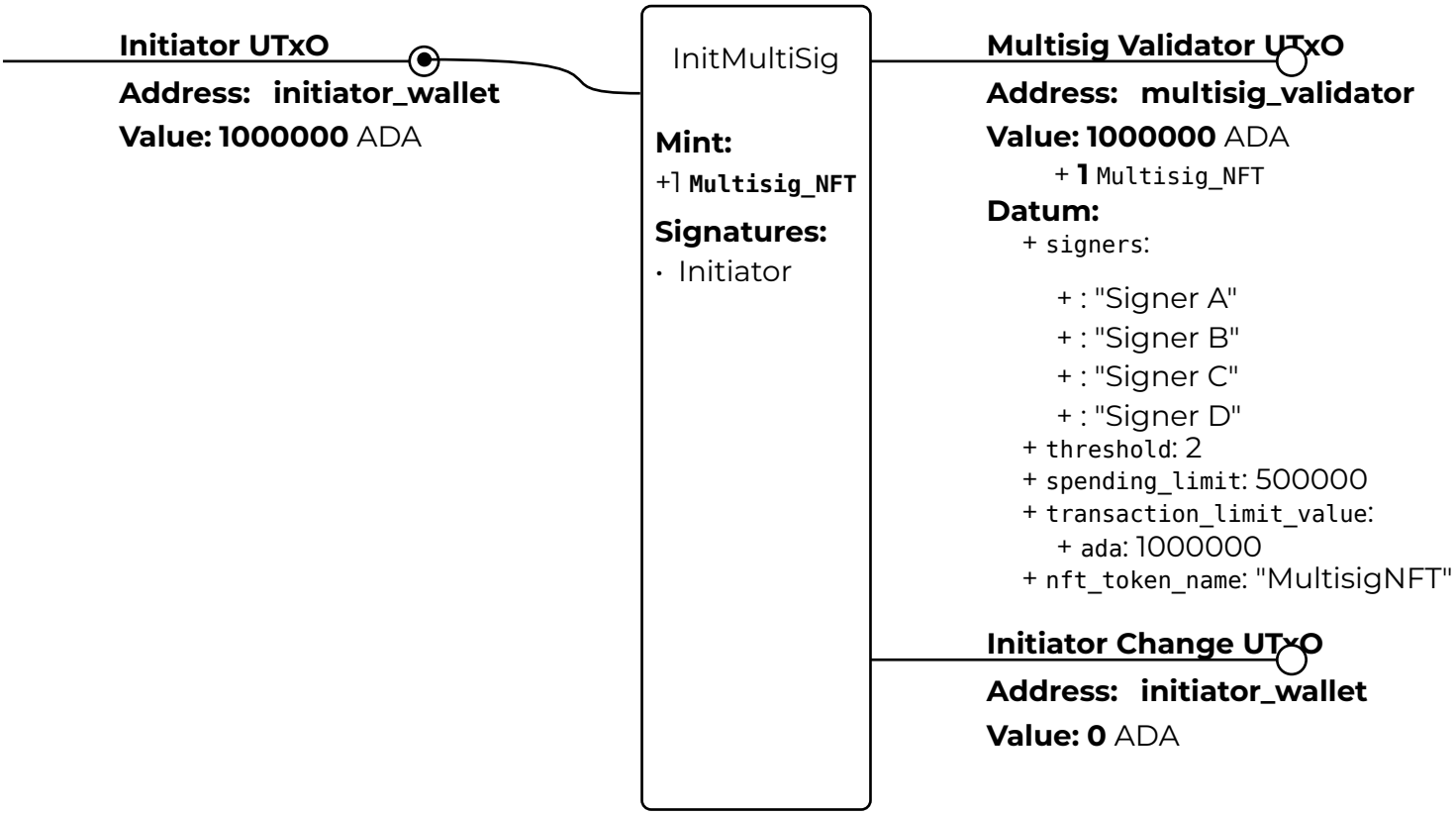
   - Verifies that the required number of authorized signers have signed the transaction
   - Enforce bounds on new signers list and threshold:
     - New signer count must be greater than 0
     - New threshold must be greater than 0 and less than or equal to the new signer count
     - New spending limit must be greater than or equal to 0 and less than or equal to the new funds quantity
     - Ensure there are no duplicate keys in the new list of signers
   - Validates that the Multisig NFT remains at the script address in the output.
   - Verify that input value equals output value (no spending during update)
   - Verify that the total value is preserved (input value equals output value, no funds are spent during update)
   - Ensure the new configuration is stored in the output datum

# 4. Transactions

This section outlines the various transactions involved in the Upgradable Multi-Signature Contract on the Cardano blockchain.

## 4.1. Mint :: InitMultiSig

This transaction initializes the multi-sig contract by minting the Multisig NFT and setting up the initial configuration.




**Note**: Initiate MultiSig Transaction

### 4.1.1. Inputs

1. **Initiator Wallet UTxO**
   - Address: Initiator's wallet address

- Value:
  - Minimum ADA required
  - Any additional ADA required for the transaction

#### 4.1.2. Mints

1. **Multi-Sig Validator**
   - Redeemer: InitMultiSig
   - Value:
     - +1 Multisig NFT Asset

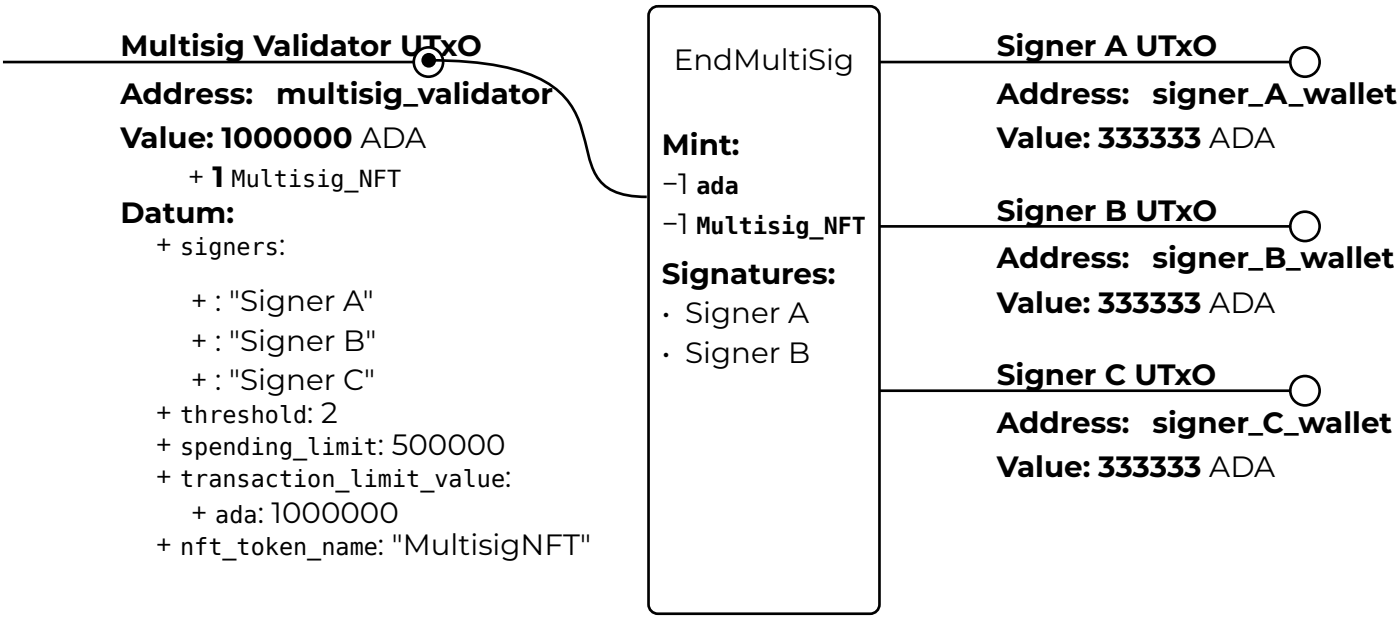
#### 4.1.3. Outputs

1. **Multi-Sig Validator UTxO**
   - Address: Multi-Sig Validator script address
   - Datum:
     - signers: 4.
     - threshold: 2.
     - spending_limit: 500,000 ADA.
     - transaction_limit_value: Asset class and quantity defining the limit for transactions.
   - Value:
     - Minimum ADA required
     - 1 Multisig NFT Asset

1. **Initiator Change UTxO** (optional)
   - Address: Initiator's wallet address
   - Value:
     - Remaining ADA after transaction fees

## 4.2. Mint :: EndMultiSig

This transaction terminates the multi-sig contract by burning the Multisig NFT and distributing the remaining funds.




**Note**: End MultiSig Transaction

### 4.2.1. Inputs

1. **Multi-Sig Validator UTxO**
   - Address: Multi-Sig Validator script address
   - Datum:
     - Current multi-sig configuration
   - Value:
     - Minimum ADA
     - 1 Multisig NFT Asset
     - Any remaining funds managed by the contract

2. **Signers' Signatures**

   Required number of signers (as per threshold) must sign the transaction.

#### 4.2.2. Mints

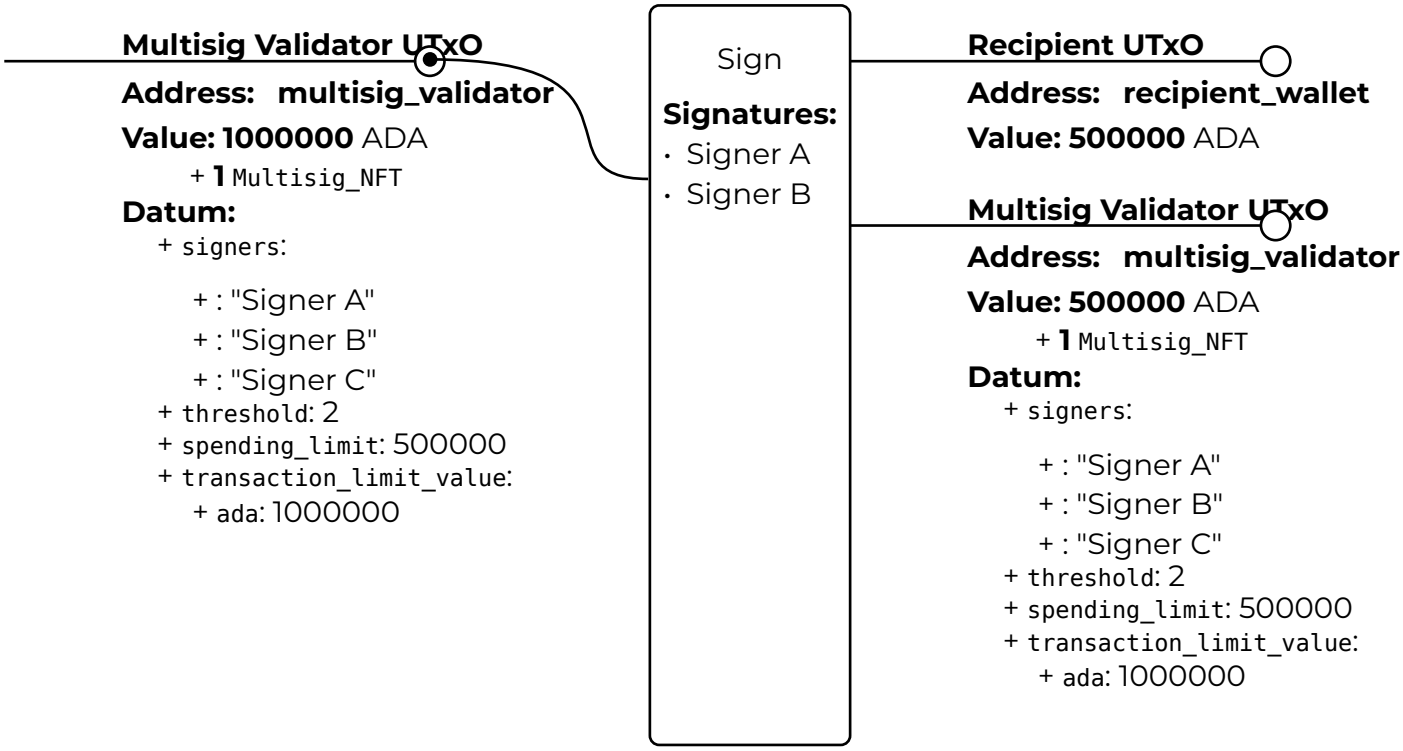
1. **Multi-Sig Validator**
   - Redeemer: EndMultiSig
   - Value:
   - −1 Multisig NFT Asset (burning the NFT)

#### 4.2.3. Outputs

1. Distribution UTxOs
   - Funds are distributed to the appropriate addresses as per the termination plan.
   - Value:
     - ADA and other assets as needed

## 4.3. Spend :: Sign

This transaction ensures that the number of signers meets or exceeds the specified threshold and The datum of the Multisig remains the same.




**Note**: Sign Transaction

#### 4.3.0.1. Inputs

1. **Multisig Validator UTxO**
   - Address: Multisig validator script address
   - Datum:
     - `signers`
     - `threshold`
     - `funds`
     - `spending_limit`

.

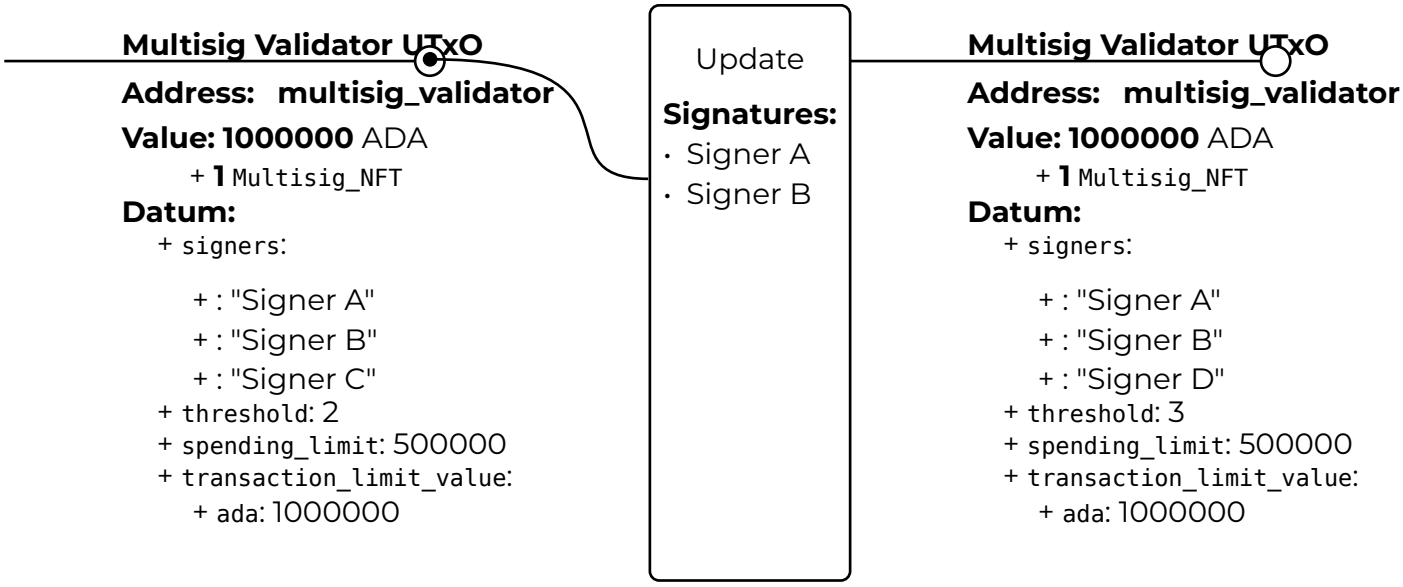
- Value:
  - ADA + Any tokens
  - Locked Value

#### 4.3.0.2. Outputs

1. **Recipient Wallet UTxO**
   - Address: Recipient wallet address
     - Transferred ADA/tokens
2. **Multisig Validator UTxO:**
   - Address: Multisig validator script address
   - Datum:
     - `signers`
     - `threshold`
     - `funds`
     - `spending_limit`
   - Value:
     - Remaining ADA + Remaining tokens

#### 4.3.1. Spend :: Update

Allows for the addition or removal of members from the Multisig arrangement, and updates the required signers threshold.




**Note**: Update Multisig Configuration

##### 4.3.1.1. Inputs

1. **Multisig Validator UTxO**
   - Address: Multisig validator script address
   - Datum:
     - `old_signers`
     - `old_threshold`
     - `funds`
     - `old_spending_limit`
   - Value:
     - X ADA + Any tokens

#### 4.3.1.2. Outputs

1. **Multisig Wallet UTxO**
   - Address: Merchant wallet address
   - Datum:
     - `new_signers`
     - `new_threshold`
     - `funds`
     - `new_spending_limit`
   - Value:
     - X ADA + Any tokens (unchanged)