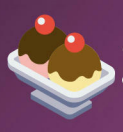
2023-02-24

Mehen

Fiat-backed Stablecoin

π Lanningham



Sundae Labs

Contents

1 - Audit Manifest

Please find below the list of pinned software dependencies and files that were covered by the audit.

Software	Version	Commit
Mehen Stablecoin	`1.0.0`	`fe63eae1cab7f42d1ebb81b9908df827240bc688`
Aiken Compiler	`aiken v1.0.24-alpha+982eff4`	`982eff449e02c0346e3db66223d983c90cd6ee9c`

Filename	Hash (SHA256)
deno/validators/usdm_control_validator.ak	`054c3b13ad5fb5d60bc106ecf411a444415974902ca1330310ef2fb0ec312695`
.../usdm_count_validator.ak	`0b8de8338df75b9064f6d8d137b42a691e5704489225f87589bf65f7392a3241`
.../usdm_minting_policy.ak	`b2d5e4e3170a8aacc5743bab696e0c09a8cec69bd7fac676ab7f05232e3ab9fe`
.../usdm_token_counter_..._nft_mint.ak	`41429696a31fccad078830c182fec39d0df34ec5526df3420fdb4ac132f81843`
deno/lib/mehen/helpers.ak	`d5435b580dc4e1e781f73f00955d57c7a54098c8070d691c7714e4d6bf26f623`
.../mehen/types.ak	`a2acf082a2e4def3e7bc05016344ede076ac8660773deac6789801ac104995f0`

Parameter	Value
`control_delay`	`1209600000`
`count_delay`	`1209600000`
`utxo_ref`	`e87493093d5c5166a43b9c920ca3d6baf8a6a7407a4f28a18bb073c93ae49ee1#1`
`c3_nft`	`73105a3662d59e002a71c12bc0017778210f4a13c54e39590cf6fc8b.4f7261636c6546656564`
`token`	`c48cbb3d5e57ed56e276bc45f99ab39abe94e6cd7ac39fb402da47ad.0014df105553444d`

Validator	Method	Hash (Blake2b-224)
usdm_token_counter_and_con-	mint_NFTs	`e319d8e6629ff7991c8ae4f8aec2e0f10463ebdf29b57d26d34914f6`
usdm_control_validator	spend	`6b3cb86f8aee9b5d754ba7aff3acea986297e833d52af0b00efe37c7`
usdm_count_validator	update_counter	`3e7530a023f5780a8fb800d23f9a338b76e5c4f2dd2edd19b29610fb`
usdm_minting_policy	mint	`c48cbb3d5e57ed56e276bc45f99ab39abe94e6cd7ac39fb402da47ad`

2 - Specification

The first part of any audit that Sundae Labs performs involves developing a deep and intimate understanding for what the client is trying to accomplish.

We first work with the client to develop a clear high level mission statement capturing the business goals of the project. Then, we translate that into an informal specification, which loosely outlines how to achieve that goal. Finally, we translate that, often with the help of the code they've written, into a detailed specification of how the protocol works.

As we make recommendations and findings, we update this specification.

We present the final version of each of these as part of our audit report.

2.a - High Level Objectives

1. The Mehen Protocol seeks to facilitate the on-chain tokenization of real-world assets.
2. By the nature of real-world assets, there are some aspects of centralization and trust that are seemingly unavoidable.
3. Instead, the Mehen protocol seeks to establish a set of checks and balances, and serve as a "decentralized watchdog" for these centralized aspects, such that efforts to tamper or manipulate these protections are thwarted or visible to those watching on-chain.
4. In particular, for some Mehen token X, ensure that, up to the reliability of some independent oracle mechanism, the amount of minted token X is equal to the value of a real world asset reported by the oracle. Such an oracle, for example, could attest to the balance of a bank holding USD.

2.b - Informal Specification

1. The "Mehen protocol" consists of one or more sub-protocols, each of which allow minting of some token representing a single "real-world backed asset".
    1. Each sub-protocol consists of a token name, an oracle, an owner, a minter, and a distributor.
    2. The oracle can publish updates to an on-chain value, which represents an attestation to a real-world ownership of some asset by the `owner` of the sub-protocol.
    3. The owner of a specific Mehen instance can upgrade the protocol, including switching to another oracle instance.
        1. For example, the initial deployment will use Charli-3, but the owner may switch to a newer version of Charli-3, a different oracle, or a combination of multiple oracles.
        2. The integration mechanism with Charli-3 is covered by the audit, but Charli-3 itself is not covered by the audit.

4. The Minter is authorized to mint or burn an on-chain token with a unique policy ID and the specified token name, representing the real world asset.
5. The tokens minted must be paid to the distributor.
    1. The distributor is a normal wallet address that is a custodian, and distributes according to their accounting of real-world deposits, transfers, withdrawals, etc.
    2. Authorization and correctness of this distribution process is outside the scope of protocol and the audit.
6. The minting contract ensures that, assuming the oracle is honest and accurate, the token is never minted unless it would result in a token supply less than or equal to the amount attested by the oracle.
    1. Specifically, the oracle balance may drop below the total supply. The protocol does not have an automated mechanism to reduce the supply, but new issuance of token is disabled until the oracle reports a higher value than the circulating supply.
7. The minting contract ensures that the `minter` of the protocol is the only one allowed to mint or burn the tokens.

2.c - Detailed

2.c.i - Definitions

- **Mehen Protocol Instance**
  - A specific instantiation of the Mehen protocol, under control of a specific **Owner** , subject to a specific **Oracle** , and regulating the supply of a specific **Mehen Asset**

- **Mehen Asset**
  - A specific `(policy_id, asset_name)` cardano native token minted and burned by a specific **Mehen Protocol Instance**

- **Owner**
  - The actor responsible for a specific **Mehen Protocol Instance** , with the ability to upgrade the **Mehen Protocol Instance** after a **Time Delay** or change certain settings. Represented on chain by as a **Multisig Script**

- **Minter**
  - The actor responsible for minting and burning a specific **Mehen Asset** . Represented on chain by a **Multisig Script**
  - The minter is separate from the **Owner** to minimize the risk of a compromised **Owner** key.

- **Distributor**
  - An address to which a minted **Mehen Asset** must be paid, for further distribution to final owners.
  - The distributor is separate from the **Owner** to minimize the risk of a compromised **Owner** key.

- **Circulating Supply**
  - The sum of all **Mehen Asset** across all UTXOs as of a specific block height on the Cardano blockchain.
  - Alternatively, the sum of all minted **Mehen Asset** tokens minus the sum of all burned **Mehen Asset** tokens.

- **Oracle**

- A process or actor which produces, on a regular basis, an attestation of some numeric value in the form of a field on an authenticatable UTXO. This process is assumed to be secured, trusted, and independent from the protocol.

- **Charli3**
  - A particular instance of an **Oracle** that the initial **Mehen Protocol Instance** will deploy with.

- **Control NFT**
  - A non-fungible token that authenticates a singleton instance of the **Mehen v1 Control Validator** or **Future Mehen Protocol Scripts** .

- **Counter NFT**
  - A non-fungible token that authenticates a singleton instance of the **Mehen v1 Count Validator** or **Future Mehen Protocol Scripts** .

- **Mehen v1 Control Validator**
  - One of the smart contracts covered by this audit, responsible for mediating changes by the **Owner** to the **Mehen Protocol Instance** settings.

- **Mehen v1 Count Validator**
  - One of the smart contracts covered by this audit, responsible for tracking and regulating the circulating supply of a given **Mehen Asset** minted by the **Minter** .

- **Control UTXO**
  - The current UTXO at any time holding the **Control NFT** , usually locked by the **Mehen v1 Control Validator** .

- **Counter UTXO**
  - The current UTXO at any time holding the **Counter NFT** , usually locked by the **Mehen v1 Count Validator** .

- **Mehen v1 Minting Policy**
  - One of the smart contracts covered by this audit, responsible for mediating the minting of a specific **Mehen Asset** .

- **Mehen Instance NFT Minting Policy**
  - One of the smart contracts covered by this audit, responsible for creating two unique NFTs for each **Mehen Protocol Instance**

- **Future Mehen Protocol Scripts**
  - A future version of the scripts making up the Mehen protocol, into which a **Owner** might upgrade to after some discernible **Time Delay**

- **Time Delay**
  - A specific and hard coded delay enforced by the **Mehen v1 Control Validator** or the **Mehen v1 Count Validator** before the **Owner** can enact any changes to the **Mehen Protocol Instance** , such as triggering an **Upgrade** , by paying one of the respective NFTs into a new contract.
    In theory, there are separate delay parameters for the **Mehen v1 Control Validator** and the **Mehen v1 Count Validator** .

- **Upgrade**
  - The process by which a **Owner** can upgrade the **Mehen Protocol Instance** to a new set of contracts after a **Time Delay**

- **Reclaim Timer**
  - The on-chain status of an **Upgrade**

- **Destination**
  - The address and datum to which a successful **Upgrade** must pay the relevant upgraded token, stored in the relevant datum for the duration of the **Time Delay** .

- **Cancellation**
  - The cancellation of a pending **Upgrade** by the **Owner**

- **Burned Bridge**
  - An emergency and last resort safety mechanism available to an **Owner** who believes the **Owner** key has been compromised, which disables all future upgrade actions, effectively locking the protocol at a specific version.

This allows the protocol to plan for a graceful upgrade in the face of a compromised **Owner** key, without needing to maintain constant vigilance against a malicious upgrade action.

- **Multisig Script**
  - A set of conditions, such as a threshold count of signatures, or an on-chain script that must be run.
  - Provided by the open source library `SundaeSwap-finance/aicone`, which (due to conflict of interest), is not covered by this audit.

- **Input Datum**
  - The datum attached to the UTXO being spent.

- **Output Datum**
  - The datum attached to the output with the appropriate authenticating NFT, passed through from the UTXO being spent.

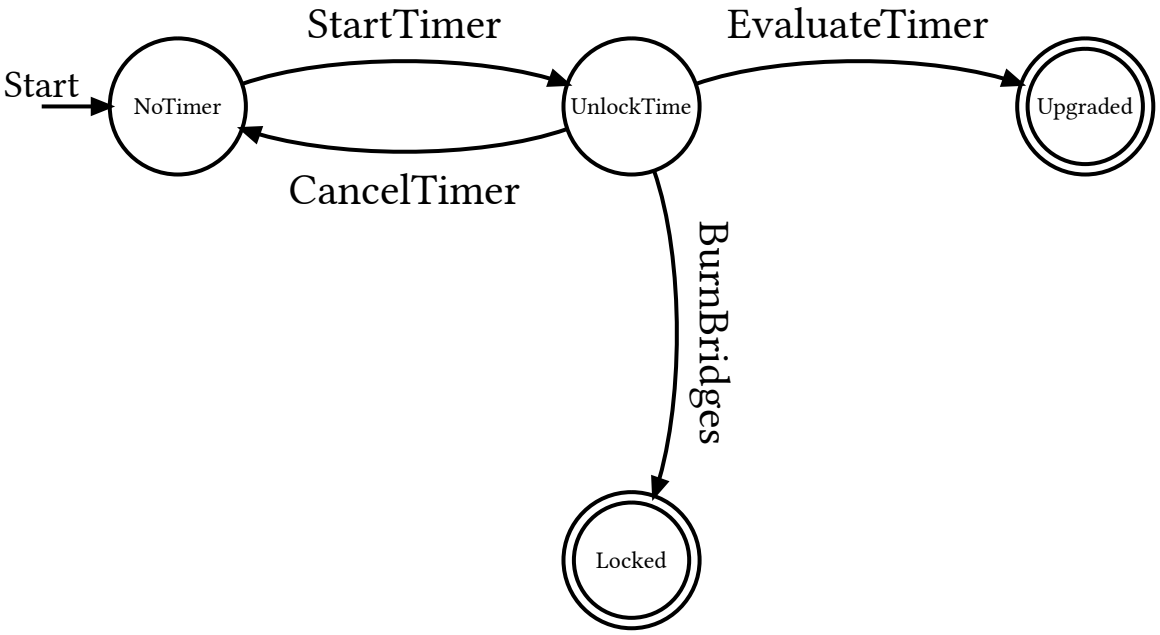
- **Control Datum**
  - The datum attached to the reference input containing the **Control NFT**

- **POSIX Time**
  - A timestamp represented as milliseconds since the UNIX epoch start, 12:00 UTC on January 1st, 1970

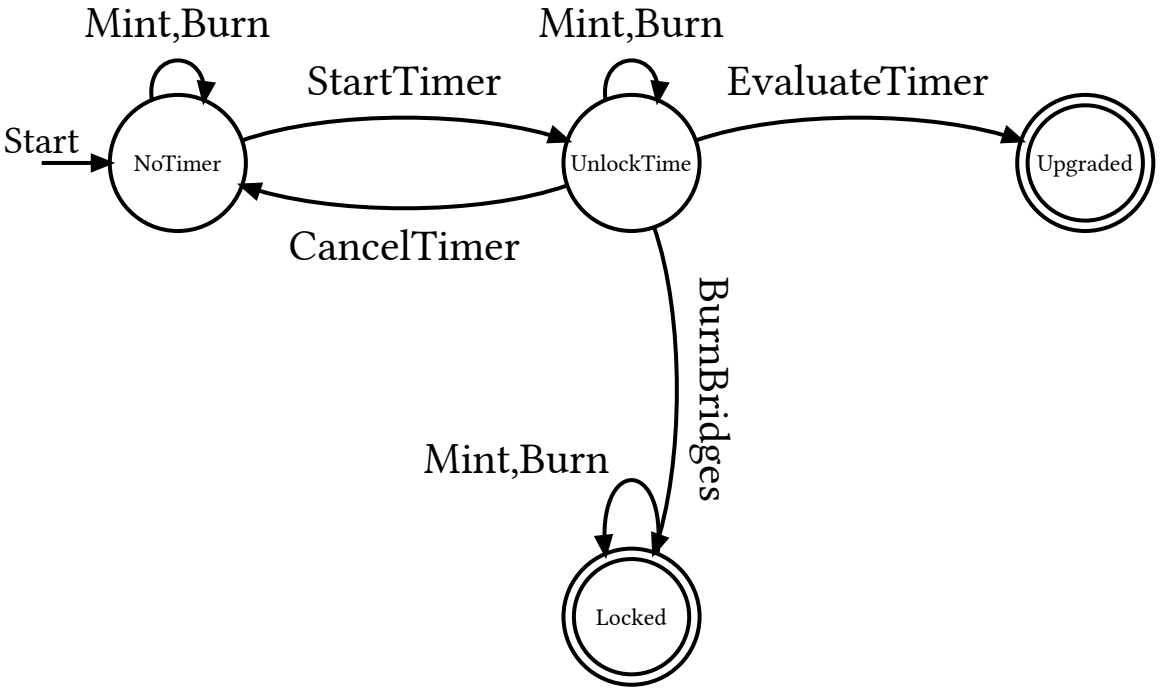
2.d - State Machine

Before digging into each specific transaction, the two relevant UTXOs of the protocol can be seen as implementing a state machine, with each transition being a transaction.

The Control UTXO follows the following state machine:



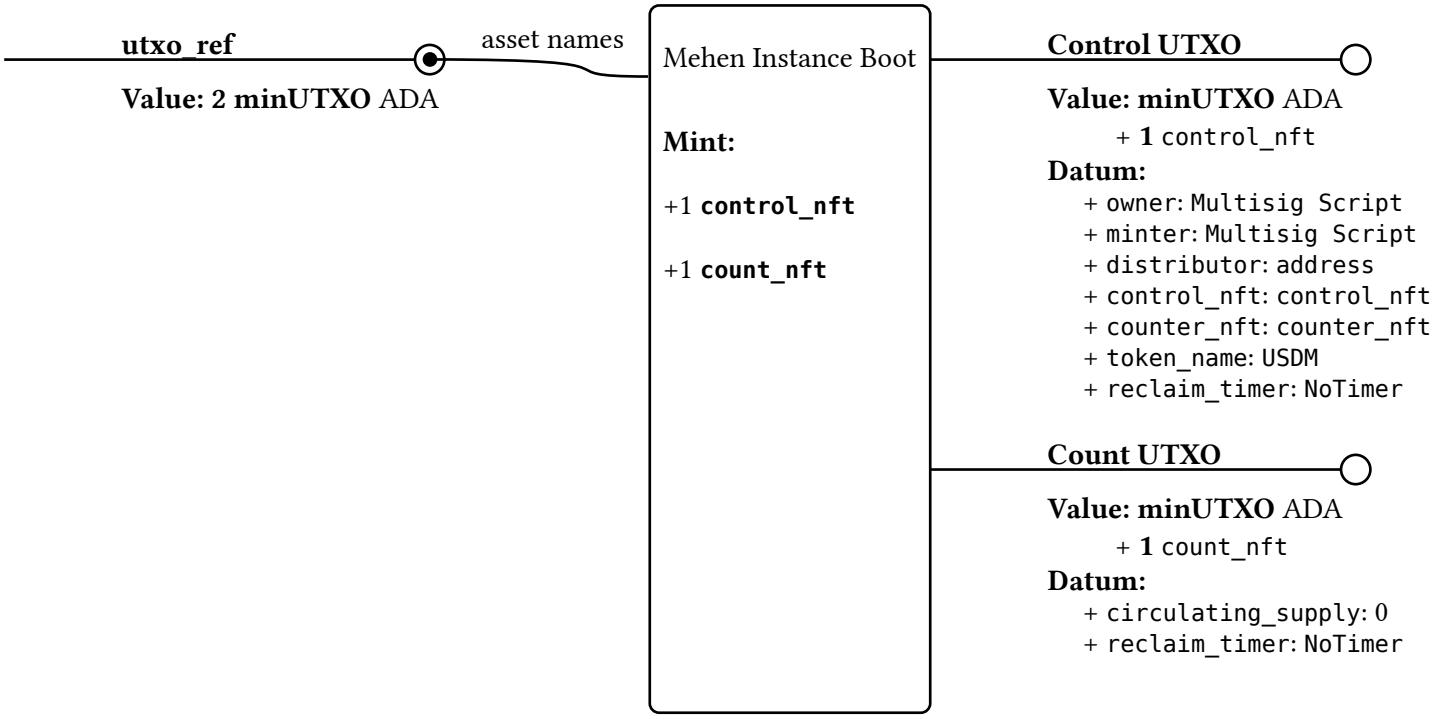
And the Count UTXO implements the following state machine:



2.d.i - Transactions

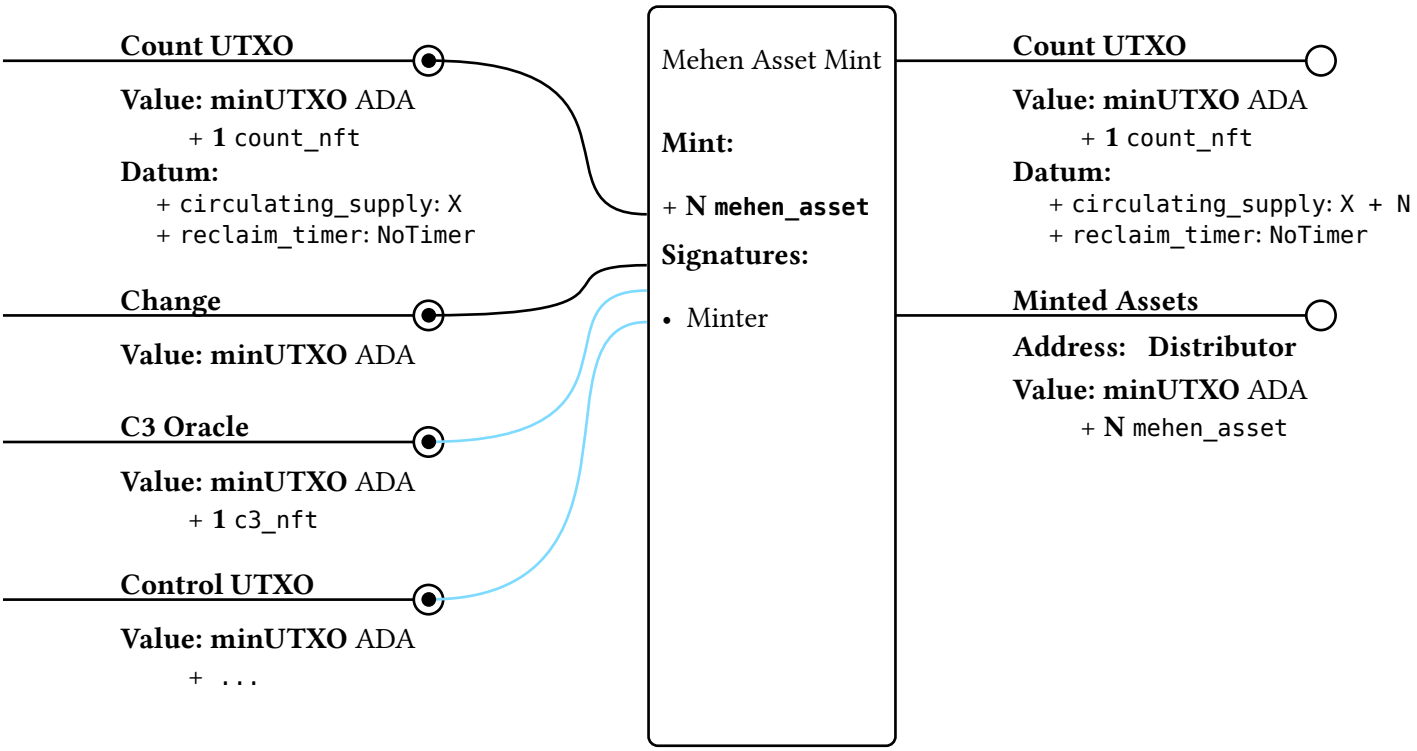
There are 7 relevant transaction archetypes. Here is a brief overview of each.

1. **Mehen Protocol Instance** Boot
    1. Mint the relevant control and count tokens via the **Mehen Instance NFT Minting Policy**
    2. The tokens may be paid into the contracts immediately, or held in a wallet and paid into the contracts later.
    3. The protocol should not be considered valid until the tokens are held by the respective contracts with a correct datum.



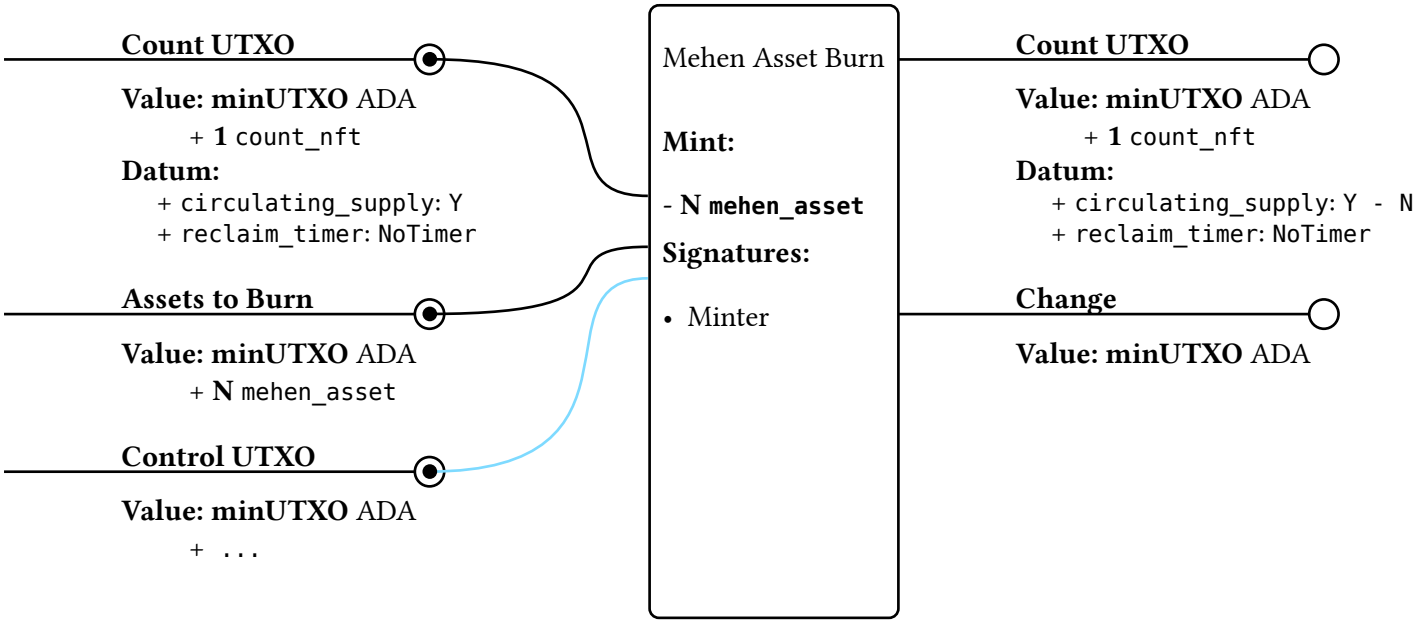
Note: `utxo_ref` must be spent

1. **Mehen Asset** Mint
    1. Mint a specific quantity of the **Mehen Asset** , paid to the distributor, increasing the **Counter UTXO** circulating supply.
    2. Must satisfy the `minter` condition from the **Control Datum**
    3. The minting policy should ensure that the new circulating supply is less than or equal to the value reported by the **Charli3** oracle.
    4. The **Charli3** oracle must have been updated recently.

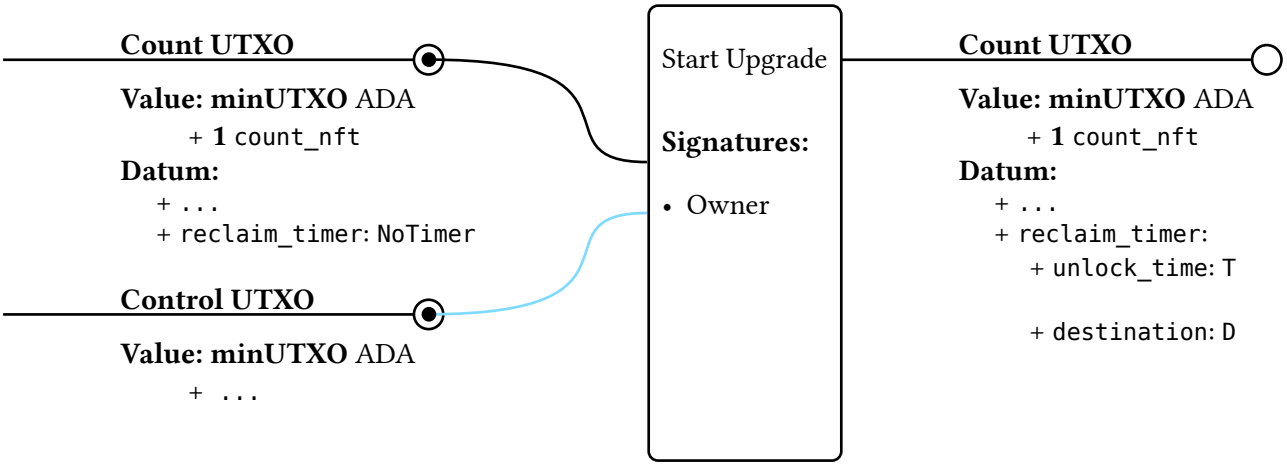


Note: C3 Oracle must have a recent timestamp

1. **Mehen Asset** Burn
    1. Burn some amount of the **Mehen Asset** , decreasing the **Counter UTXO** circulating supply.
    2. Must satisfy the `minter` condition from the **Control Datum**

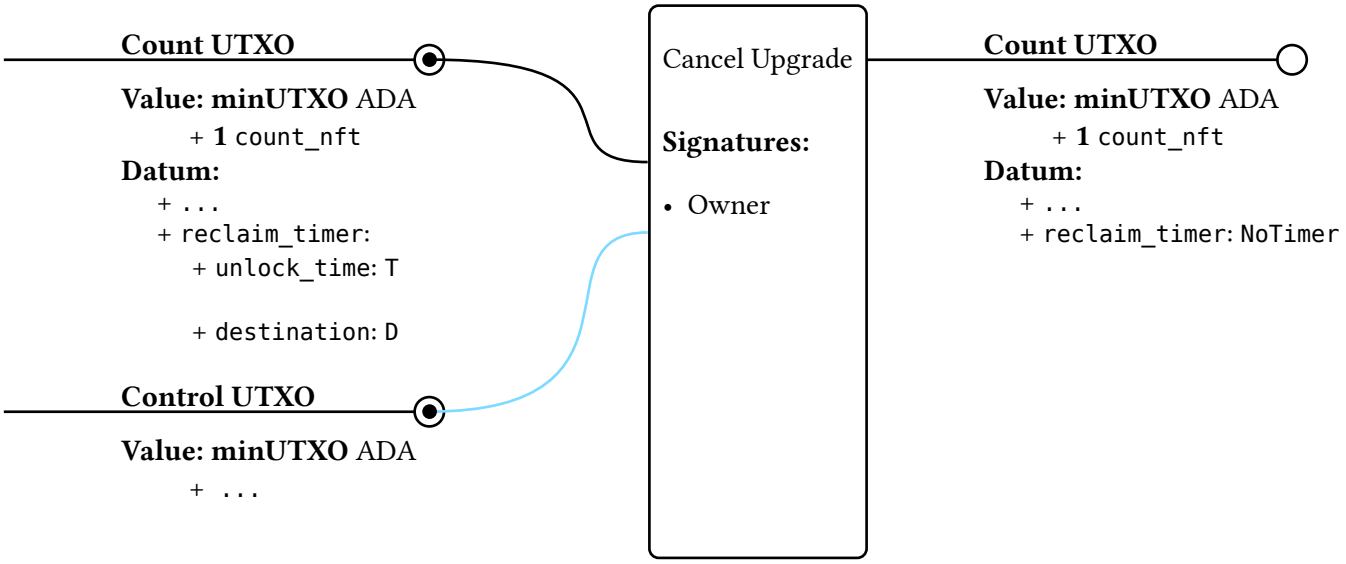


1. **Mehen Protocol Instance** Start Upgrade
    1. Initiate a **Time Delay Upgrade**
    2. Applicable to either the **Counter UTXO** or the **Control UTXO** . We present only one of them below.



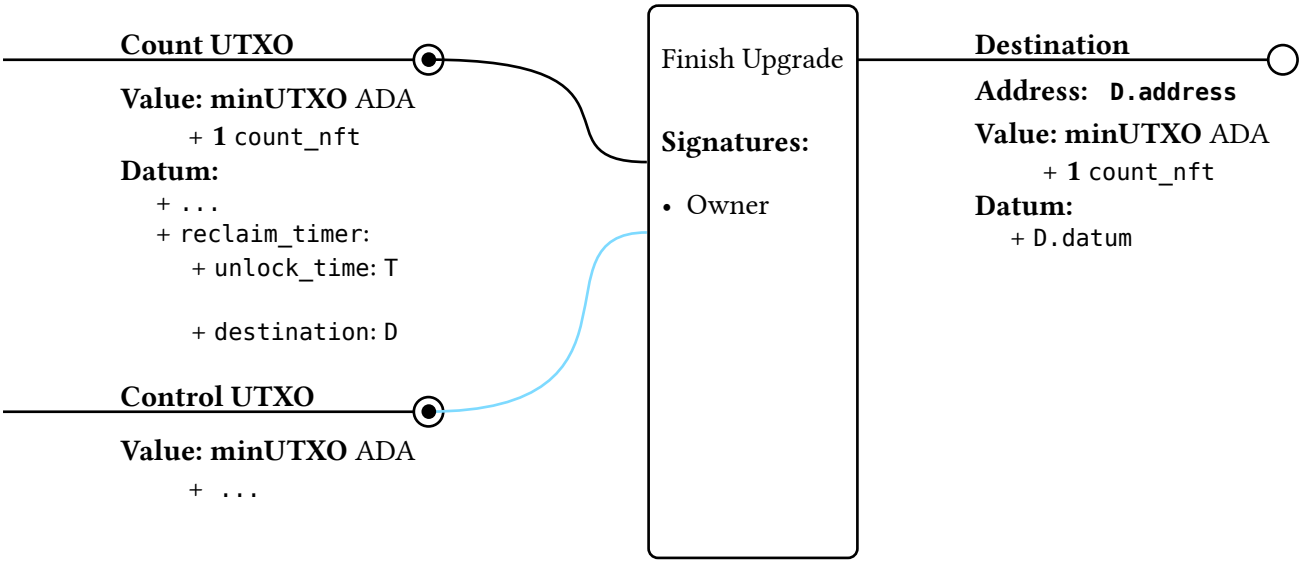
Note: `T` must be `time_delay` after the transaction upper bound

1. **Mehen Protocol Instance** Cancel Upgrade
    1. Cancel a pending **Time Delay Upgrade**
    2. Applicable to either the **Counter UTXO** or the **Control UTXO** . We present only one of them below.



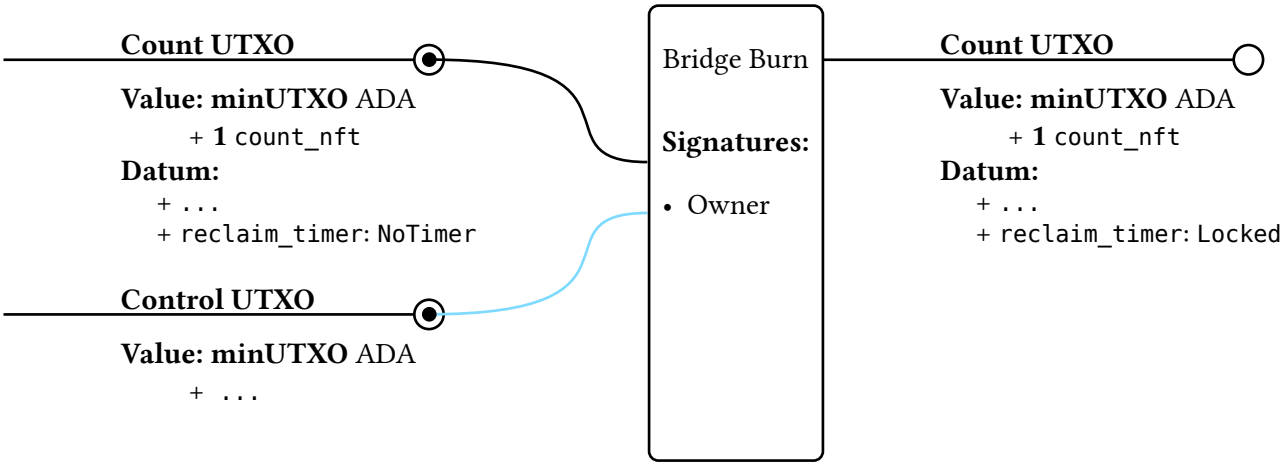
Note: `T` must be after the transaction upper bound

1. **Mehen Protocol Instance** Finish Upgrade
    1. Finish a pending **Time Delay Upgrade**
    2. Applicable to either the **Counter UTXO** or the **Control UTXO** . We present only one of them below.



Note: `T` must be before the transaction lower bound

1. **Mehen Protocol Instance** Bridge Burn
    1. Permanently prevent all future **Time Delay Upgrade**
    2. Applicable to either the **Counter UTXO** or the **Control UTXO** . We present only one of them below.



2.d.ii - Requirements

We now provide a detailed, point by point specification of each of the 4 smart contracts covered by this audit.

1. Initialization
    1. A **Mehen Protocol Instance** is instantiated by minting two unique NFTs, the **Control NFT** and the **Counter NFT** .
    2. The protocol is not considered valid until there is no possibility of minting new versions of these assets; For the purposes of this audit, this is handled by the **Mehen Instance NFT Minting Policy**
    3. The protocol is not considered valid until these NFTs are held by their respective contracts with the correct datum, and the transaction which does so is at least 2160 blocks behind the current tip (the "rollback horizon" for Cardano).
    4. The **Control NFT** must be paid to the **Mehen v1 Control Validator**
        1. The datum of this UTXO must be initialized with:
            1. The **Owner** of the protocol, a valid **Multisig Script**
            2. The **Minter** for the protocol, a valid **Multisig Script**
            3. The **Distributor** for the protocol, a Cardano address
            4. The correct `policy_id` and `asset_name` of the **Control NFT**
            5. The correct `policy_id` and `asset_name` of the **Counter NFT**
            6. The on-chain asset name for the **Mehen Asset** for this protocol instance
            7. The `reclaim_timer` set to `NoTimer`
    5. The **Counter NFT** must be paid to the **Mehen v1 Count Validator**
        1. The datum of this UTXO must be initialized with:
            1. A `circulating_supply` of 0
            2. The `reclaim_timer` set to `NoTimer`
    6. The actual on-chain circulating supply of the **Mehen Asset** must be 0 at the time that the **Counter NFT** is locked in the **Mehen v1 Count Validator**
    7. A single "CIP-68" reference token for the **Mehen Asset** may be minted.
    8. The `c3_nft` validator parameter must refer to an asset with a provable, permanent quantity of 1, and be locked by the **Charli3** protocol.

1. The validity and trustworthiness of the **Charli3** protocol was not covered by this audit, and is assumed to be secure and reliable.
2. The `c3_nft` asset must be locked in a UTXO with the following datum:
    1. A single `info` property, of type `Info`
    2. The `Info` type has multiple constructors, but only constructor 2 is relevant to this protocol
    3. Constructor 2 consists of a single dictionary property
        1. Key 0 of the dictionary is set to the published oracle supply value
        2. Key 1 of the dictionary is set to the timestamp of the published value, as a **POSIX Time**
        3. Key 2 of the dictionary is set to the timestamp when the value expires, as a **POSIX Time**
2. The **Mehen Instance NFT Minting Policy** controls the minting of the **Control NFT** and the **Counter NFT**.
    1. The script is parameterized by `utxo_ref`, a transaction ID and output index that must be spent as an input on the transaction.
    2. The script can be run as a minting policy, meaning it takes a redeemer and a script context.
    3. The redeemer has one constructor with two fields, `counter_name` and `control_nft_name`, both `AssetNames`, an alias for `ByteArray`
    4. The script succeeds if all of the following conditions are met:
        1. The script context `purpose` is `Mint`; Let `own_policy_id` be the policy ID in the script purpose. The Cardano blockchain ensures this is the minting policy of the assets being minted.
        2. The transaction `mint` field is a value with a quantity of 1 for `(own_policy_id, counter_name)`
        3. The transaction `mint` field is a value with a quantity of 1 for `(own_policy_id, control_nft_name)`
        4. At least one of the spent transaction inputs has an output reference equal to the `utxo_ref` parameter.
    5. Note that this never allows the assets to be burned, or minted a second time.
3. The **Mehen v1 Control Validator** controls global protocol settings, such as the owner and minter.
    1. The script is parameterized by a `control_delay` property, which is an integer that represents a number of milliseconds.
    2. The script can be run only as a spending validator, meaning it takes an existing datum, a redeemer, and a script context.

3. The redeemer has one constructor with an `action` field.
4. The `action` is of type WithdrawableRedeemerAction, and has 4 constructors:
    1. `StartTimer`, to initiate a change to the protocol, with a `destination` field, which consists of an `Address` and a `Datum`.
    2. `CancelTimer`, to cancel a pending change to the protocol
    3. `EvaluateTimer`, to enact the pending change when the timer has expired
    4. `BurnBridges`, to permanently prevent all future pending changes
5. The script succeeds, allowing the **Control UTXO** to be spent, if all of the following conditions are met:
    1. Let `control_nft` be the **Control NFT** specified in `datum.control_nft`
    2. There must be at least one output on the transaction with a value such that `quantity_of(datum.control_nft.policy_id, datum.control_nft.asset_name) == 1`
        1. The datum on this output must be either an InlineDatum, or a DatumHash with a corresponding datum in the transaction datum witnesses
        2. The contents of this datum must be of type `ControlDatum`
        3. Let the contents of the datum attached to the output described above be called `control_output_datum`
        4. n.b. in the code, this output and datum are identified in each branch, rather than before the branching, but in each case it is the same.
    3. When `redeemer.action` is `StartTimer`, with a `destination`, then all of the following conditions must be met:
        1. `datum.reclaim_timer` must be `NoTimer`
        2. `control_output_datum.reclaim_timer` must be valid, as defined below
            1. The transaction upper bound must be finite; Let this upper bound be `tx_upper_bound`
            2. Let `final_delay` be the transaction upper bound, plus the `control_delay` parameter.
            3. The `control_output_datum.reclaim_timer` must be `UnlockTime`.
            4. The transaction mint field must be the zero value.
            5. The `datum.owner` **Multisig Script** must be satisfied. See **MEHEN-301**
            6. `control_output_datum.reclaim_timer.unlock_time` must be greater than `final_delay`

7. `control_output_datum.reclaim_timer.unlock_time` must be less than `final_delay + 3600000`, which is 1 hour after `final_delay`.
8. `control_output_datum.reclaim_timer.destination` must be equal to the `destination` field in the `redeemer`
9. The **Control NFT** must be included in an output with the same address as the input containing the **Control NFT**

3. All fields on `control_output_datum` except `reclaim_timer` must equal the values from the input `datum`

4. When `redeemer.action` is `CancelTimer`, then all of the following conditions must be met:
   1. `datum.reclaim_timer` must be `UnlockTime`
   2. `control_output_datum.reclaim_timer` must be `NoTimer`
   3. The transaction mint field must be the zero value.
   4. The `datum.owner` **Multisig Script** must be satisfied. See **MEHEN-301**
   5. The **Control NFT** must be included in an output with the same address as the input containing the **Control NFT**
   6. All fields on `control_output_datum` except `reclaim_timer` must equal the values from the input `datum`
5. When `redeemer.action` is `EvaluateTimer`, then all of the following conditions must be met:
   1. If the output that contains the **Control NFT** has the same address as the input, then the `control_output_datum.reclaim_timer` must be `NoTimer`
   2. The input `datum.reclaim_timer` must be `UnlockTime`
   3. The transaction mint field must be the zero value.
   4. The `datum.owner` **Multisig Script** must be satisfied. See **MEHEN-301**
   5. The output containing the **Control NFT** must have an address equal to `datum.reclaim_timer.destination.address`, and a datum equal to `datum.reclaim_timer.destination.datum`
   6. The transaction lower bound must be finite, and must be greater than `datum.reclaim_timer.unlock_time`
6. When `redeemer.action` is `BurnBridges`, then all of the following conditions must be met:
   1. `datum.reclaim_timer` must be `UnlockTime` (n.b. this isn't strictly neccesary)

2. `control_output_datum.reclaim_timer` must be `Locked`
3. The transaction mint field must be the zero value.
4. The `datum.owner` **Multisig Script** must be satisfied. See **MEHEN-301**
5. The **Control NFT** must be included in an output with the same address as the input containing the **Control NFT**
6. All fields on `control_output_datum` except `reclaim_timer` must equal the values from the input `datum`

4. The **Mehen v1 Minting Policy** allows minting of the **Mehen Asset**
   1. The script is parameterised by `counter_nft`, the **Counter NFT** minted on protocol instantiation
   2. The script can be run only as a minting policy, meaning it takes a redeemer and a script context.
   3. The script allows minting if there is at least one transaction output such that the value has a `quantity_of(counter_nft.policy_id, counter_nft.asset_name)` equal to 1.
5. The **Mehen v1 Count Validator** controls the minting and burning of the **Mehen Asset** , ensuring minting is disallowed if it would bring the circulating supply above the oracle reported value.
   1. The script is parameterized by the following parameters:
      1. `token`, the **Mehen Asset** to be minted or burned
      2. `control_nft`, the asset ID of the **Control NFT**
      3. `c3_nft`, the asset ID of the **Charli3** NFT
      4. `count_delay`, the minimum delay needed to enact a protocol change to the counting script
   2. The script can only be run as a spending validator, meaning it takes a datum, a redeemer, and a script context.
   3. The redeemer has 3 constructors:
      1. `Mint`, to mint new **Mehen Asset**
      2. `Burn`, to burn **Mehen Asset**
      3. `WithdrawCount`, which has an action field of type `WithdrawableRedeemerAction`, as used in the **Mehen v1 Control Validator**
   4. The script succeeds, allowing the **Counter UTXO** to be spent, if all of the following conditions are met:
      1. Let `datum` refer to the datum attached to the **Counter UTXO** being spent

2. There must be at least one reference input on the transaction with a value such that `quantity_of(control_nft.policy_id, control_nft.asset_name) == 1`
    1. The datum on this reference input must be either an InlineDatum, or a DatumHash with a corresponding datum in the transaction datum witnesses
    2. The contents of this datum must be of type `ControlDatum`
    3. Let the contents of the datum attached to the output described above be called `control_ref_datum`
3. Unless `redeemer` is `WithdrawCount` with an `action` of `EvaluateTimer`, there must be at least one transaction output such that the value has a `quantity_of(control_ref_datum.counter_nft.policy_id, control_ref_datum.counter_nft.asset_name)` equal to 1.
    1. The datum on this output must be either an InlineDatum, or a DatumHash with a corresponding datum in the transaction datum witnesses
    2. The contents of this datum must be of type `CounterDatum`
    3. Let the contents of the datum attached to the output described above be called `counter_output_datum`
4. When `redeemer` is `Mint`, all of the following conditions must be met:
    1. `datum.reclaim_timer` must equal `counter_output_datum.reclaim_timer`
    2. There must be at least one reference input on the transaction with a value such that `quantity_of(c3_nft.policy_id, c3_nft.asset_name) == 1`
        1. The datum on this reference input must be either an InlineDatum, or a DatumHash with a corresponding datum in the transaction datum witnesses
        2. The contents of this datum must be of type `C3Datum`
        3. The value of the `info` field on this datum must be the third constructor (called `ReserveDict`), which has a single field, a Dictionary from Integer to Integer.
        4. Let the value of this dictionary `c3_info_dict`
    3. `c3_info_dict` must be well formed, as specified below:
        1. `c3_info_dict` must have a key with the value `0`, let this be `c3_oracle_supply`
        2. `c3_info_dict` must have a key with the value `1`, let this be `c3_oracle_reported_time`
        3. `c3_info_dict` must have a key with the value `2`, let this be `c3_oracle_expire_time`

4. Let `mint_count` be the `quantity_of(token.policy_id, token.asset_name)` on the transaction `mint` field.
5. `mint_count` must be strictly greater than 0.
6. The transaction must mint or burn no other tokens with a policy ID equal to `token.policy_id`
7. Let `output_count` equal `counter_output_datum.circulating_supply`
8. `output_count` must be equal to `datum.circulating_supply + mint_count`
9. `c3_oracle_supply` must be greater than or equal to `output_count`.
10. The `control_ref_datum.minter` **Multisig Script** condition must be satisfied. See **MEHEN-301**
11. There must exist no transaction inputs such that `quantity_of(token.policy_id, token.asset_name) > 0` for the value of the input
12. There must be at least one output such that the `quantity_of(token.policy_id, token.asset_name)` is equal to `mint_count`, and the output `address` is equal to `control_ref_datum.distributor`
    1. n.b. Since we ensure that no **Mehen Asset** appears on the inputs, the only source of **Mehen Asset** is from the minting policy, and the use of `any` here is safe.
13. The `c3_oracle_reported_time` must be strictly less than the transaction lower bound
14. The `c3_oracle_expire_time` must be strictly greater than the transaction upper bound
15. The first input found containing `control_ref_datum.counter_nft` must have the same address as the first output found containing `control_ref_datum.counter_nft`
    1. Because `counter_nft` is unique, the `first input found` and `first output found` are equivalent to `only input found` and `only output found`

5. When `redeemer` is `Burn`, then all of the following conditions must be met:
    1. `datum.reclaim_timer` must equal `counter_output_datum.reclaim_timer`
    2. Let `burn_count` be the `quantity_of(token.policy_id, token.asset_name)` on the transaction `mint` field.
    3. `burn_count` must be strictly less than 0.
    4. The transaction must mint or burn no other tokens with a policy ID equal to `token.policy_id`

5. `counter_output_datum.circulating_supply` must be equal to `datum.circulating_supply + burn_count`
6. The `control_ref_datum.minter` **Multisig Script** condition must be satisfied. See **MEHEN-301**
7. The first input found containing `control_ref_datum.counter_nft` must have the same address as the first output found containing `control_ref_datum.counter_nft`
    1. Because `counter_nft` is unique, the `first input found` and `first output found` are equivalent to `only input found` and `only output found`
6. When `redeemer` is `WithdrawCount`, then the logic from the **Mehen v1 Control Validator** is reused, with the following substitutions:
    1. Instead of `control_output_datum`, we use `counter_output_datum`
    2. Instead of reading from the control datum on the input for the minter, we read from the control datum as a reference input
    3. Instead of `control_delay`, we use `count_delay`
    4. Instead of checking that the `control_nft` is paid back to the script, check that the `counter_nft` is being paid back to the script
    5. Instead of comparing datum fields from the `control_output_datum` to the input `datum`, we compare the `circulating_supply` field from the `counter_output_datum` to the input `datum`

3 - Findings Summary

ID	Title	Severity	Status
MEHEN-100	No time delay on critical protocol changes	Major	Resolved
MEHEN-101	Upgradability of the token	Major	Resolved
MEHEN-102	No commitment to upgrade action	Major	Resolved
MEHEN-200	No expression of liabilities	Minor	Acknowledged
MEHEN-201	Risk if theres a large gap between circulating USDM and redeemable users	Minor	Acknowledged
MEHEN-202	Lack of flexibility in the distributor	Minor	Acknowledged
MEHEN-203	Effect of accidental burned tokens	Minor	Acknowledged
MEHEN-204	Operational weakness of single signing keys	Minor	Resolved
MEHEN-205	Potential locking of protocol	Minor	Acknowledged
MEHEN-206	Potential abuse of BurnBridges	Minor	Resolved
MEHEN-300	Missing cosmetic guards on protocol boot	Info	Acknowledged
MEHEN-301	Misuse of sundae/multisig	Info	Acknowledged

MEHEN-302	Duplicated files could lead to deployment confusion	Info	Acknowledged
MEHEN-303	Potential guards through mint diffusion rate	Info	Acknowledged
MEHEN-304	Considerations for CIP-68 tokens	Info	Acknowledged

MEHEN-100 - No time delay on critical protocol changes

Category	Commit	Severity	Status
Security	bb53420773d7c6054ae0274fe39dc5715448eaa8	Major	Resolved

Description

The goal of the Mehen protocol is to serve as a "watchdog" mechanism, to keep Mehen Innovations (or any other **Owner**) honest and transparent about the real world backing. This serves to build confidence among users, as they have to rely less on the word of any individual party, and can have confidence that the **Mehen Asset** they hold is fully solvent and backed by real world assets.
However, it also has mechanisms for updating these settings, and upgrading these watchdog mechanisms. In theory, this upgrade path can be seen on chain, so it serves as a watchdog. However, an attacker with compromised **Owner** keys (or a malicious **Owner**) could upgrade the protocol oracle to one they control, publish a new maximum supply of 100 trillion, mint a large quantity of the **Mehen Asset**, and sell it on a DEX for ADA, all within the span of a few blocks.

Recommendation

Add a **Time Delay** to the **Mehen v1 Control Validator** and the **Mehen v1 Count Validator** that applies to all changes to the protocol. Allow these queued changes to be cancelled by the **Owner**. This time delay would allow Mehen Innovations to still upgrade the protocol, rotate owner keys, etc; but in the case of a malicious change, it would give Mehen Innovations the chance to cancel the change, or users to exit their position before any attacker, greatly diminishing the risk of such an attack.
We recommend a minimum of 2 weeks for this parameter.

Resolution

This issue was resolved as of commit `bb53420773d7c6054ae0274fe39dc5715448eaa8`.

MEHEN-101 - Upgradability of the token

Category	Commit	Severity	Status
Operations	bb53420773d7c6054ae0274fe39dc5715448eaa8	Major	Resolved

Description

Currently, a large portion of the logic is codified into the minting policy for the **Mehen Asset** . If Mehen Innovations ever wants to upgrade the protocol and change any of this logic, it would require a new policy ID, which would be very operationally difficult for all users who had the **Mehen Asset** in their wallet. It would create a very extended period of fragmented liquidity, and complicate Mehen Innovation's job at running the protocol.

Recommendation

Preserve all of the same logic, but move it into the **Mehen v1 Count Validator** The minting policy should exclusively check that the **Counter NFT** is in the inputs.
This means that all minting logic can be updated (subject to the **Time Delay**), without requiring a new policy ID, by reclaiming the Counter NFT and paying it into a new script.

Resolution

This issue was resolved as of commit `bb53420773d7c6054ae0274fe39dc5715448eaa8`.

MEHEN-102 - No commitment to upgrade action

Category	Commit	Severity	Status
Security	9e1f7acc00a10b7a0667598e112cdfd1fef29a27	Major	Resolved

Description

The current upgrade mechanism enforces a time delay between starting any change and enacting that change.

However, the change to enact is provided in the redeemer when actually enacting the change.

This opens the protocol up to the following attack:

- The **Owner** key is compromised, but the attacker does nothing with the keys immediately.
- The **Owner** starts a legitimate upgrade, letting users know in advance, with open source contracts as the intended destination.
- Immediately after the **Time Delay** elapses, the attacker submits a transaction which pays the **Control NFT** to a malicious script.

Recommendation

Move the `destination` field from the `EvaluateTimer` constructor to the `StartTimer` constructor. Store the `destination` in the `ReclaimTimer` in the datum. Enforce that the relevant NFTs are paid to the destination from the datum, rather than the redeemer.

Resolution

This issue was resolved as of commit `9e1f7acc00a10b7a0667598e112cdfd1fef29a27`.

MEHEN-200 - No expression of liabilities

Category	Commit	Severity	Status
Economic		Minor	Acknowledged

Description

If the oracle only attests to a bank balance, it is only reporting assets. However, solvency is a function of both assets and liabilities. If the USD holding company backing the stablecoin were to start accruing large debts, the protocol may become insolvent while appearing solvent.

Recommendation

Add a `liabilities` field to the published oracle value, and ensure that the minted **Mehen Asset** is less than or equal to the net balance.

Resolution

This issue was acknowledged by the project team with the comment:
We agree that adding this component would be a good enhancement to the protocol in certain circumstances, but it would be inappropriate for the USDM stablecoin implementation.
The USDM stablecoin is a "senior secured" obligation of Mehen Finance LLC, with its primary securing asset being the USDM Reserve accounts, and with additional recourse to all of the unencumbered corporate assets of Mehen Finance LLC. By law and according to the legal documents that govern USDM issuance, the USDM stablecoin is the only obligation which can be issued against the USDM Reserve. To that end, all "liabilities" of the USDM Reserve are expressed by the value in the "counter token" and are accounted for in this protocol. To include an additional expression of "liabilities" would be inappropriate in this context.

MEHEN-201 - Risk if theres a large gap between circulating USDM and redeemable users

Category	Commit	Severity	Status
Economic		Minor	Acknowledged

Description

Someone who buys a **Mehen Asset** such as USDM on the open market, such as off a DEX, has no KYC relationship with Mehen Innovations, and may not be able to directly redeem the token for USD. In theory, USDM still maintains its peg because there will likely be someone who is willing to buy and redeem that asset. However, if there are very few such users, with minimal access to liquidity, then outflows of USDM may be bottlenecked and the asset may lose its peg.

Recommendation

Consider publishing an economic health report that considers such metrics to bolster confidence. Consider building relationships with a diverse set of liquidity providers.

Resolution

This issue was acknowledged by the project team with the comment: We acknowledge that the ongoing ability of the token to maintain its intended peg will largely depend on the aggregate arbitrage power of the user base. This is a matter that will be addressed during the launch and maintenance of the USDM stablecoin.
The Mehen team is constantly working to expand both the number of mint/burn enabled individuals, and their aggregate arbitrage power.
We will take the dashboard suggestion under advisement and will seek appropriate ways to communicate this information. This is not a matter that can be resolved at this time by modifying the smart contracts.

MEHEN-202 - Lack of flexibility in the distributor

Category	Commit	Severity	Status
Operations		Minor	Acknowledged

Description

Right now, minted **Mehen Asset** must be paid to the distributor, which is a normal wallet address. This limits flexibility and extensibility. For example, perhaps the distributor in the future could be a multisig, a time delay contract, or a merkel-tree powered self-claim smart contract.

Recommendation

Consider making the distributor a "destination", which is an arbitrary address. To prevent accidental burns of **Mehen Asset** , if the address has a script validator for the payment credential, enforce that some datum is attached.

Resolution

This issue was acknowledged by the project team with the comment: Considering potential requirement changes.

MEHEN-203 - Effect of accidental burned tokens

Category	Commit	Severity	Status
Accidents		Minor	Acknowledged

Description

If **Mehen Asset** is accidentally burned, the backing real world asset would be permanently inaccessible. That is, even though the real circulating supply has been reduced, the real world assets can't be recapitalized, as doing so would prevent the **Mehen Asset** from being minted.
For example, if someone accidentally sends 10m USDM to a script address without a datum attached, or 10m USDM gets locked in a DeFi protocol, then in both of these cases it is provable on-chain that the USDM will never re-enter the circulating supply. However, there will be 10m real world USD that can't be used as liquidity in the Cardano ecosystem.
Mehen Innovations would be unable to work with users to unlock these funds, for example, without the cooperation of the oracle, or a new protocol version.

Recommendation

Add a property to the **Oracle** tracking the "burned" **Mehen Asset** . Factor this into the circulating supply for the purposes of regulating minting. In this way, if a user accidentally locks USDM in a provable way, and can prove ownership of the wallet, Mehen Innovations would be able to help them regain access to their funds.

Resolution

This issue was acknowledged by the project team with the comment:
We have considered this, and are open to finding a solution to this problem in the future. There is currently no reliable systematic way to find UTXOs that have "burned" "lost" or otherwise "irredeemable USDM," and building one would introduce additional risks.
Any amount of accidentally burned, lost, or otherwise irredeemable USDM will result in a "excess reserves" scenario for the USDM Reserve, which will further support the pool of redeemable USDM.
We encourage all USDM users to follow proper security and operational protocols, and carefully monitor their transactions to avoid losing their USDM.

MEHEN-204 - Operational weakness of single signing keys

Category	Commit	Severity	Status
Security	bb53420773d7c6054ae0274fe39dc5715448eaa8	Minor	Resolved

Description

The **Owner** is a single public key hash. It also must be a “hot key”, used by the production systems to mint the **Mehen Asset** . This creates an extreme risk of the key being compromised.
Similarly, the **Owner** and **Minter** responsibilities are on the same public key hash. The risk of these two keys is very different, and only the **Minter** key needs to be a “hot key”.
Finally, similar arguments apply to the use of a single VerificationKeyCredential for the **Distributor** .

Recommendation

Split the **Owner** and **Minter** responsibilities into two separate fields. Allow a richer set of conditions, such as multisignature schemes, for each of these responsibilities. We have an open source library called `SundaeSwap-finance/aicone` that provides exactly this capability. Finally, change the type of the **Distributor** field to `Address`, so it can be a multisignature script.

Resolution

This issue was resolved as of commit `bb53420773d7c6054ae0274fe39dc5715448eaa8`.

MEHEN-205 - Potential locking of protocol

Category	Commit	Severity	Status
None		Minor	Acknowledged

Description

Several of the code paths check that the NFT is paid back to the script, but they don't check that there are no other tokens added.
In subtle cases, this could allow someone who compromises the `owner` key to "pack" the UTXO with random cardano native tokens.
Done correctly, this can lock the protocol, as it becomes impossible to spend the input again, because deserializing the value would exceed execution units.

Recommendation

Because this is guarded behind the `owner` key, we consider this minor severity.
If you'd like to correct this, or correct it in a future version, the fix is to ensure that the output contains only the NFT and some amount of lovelace.

Resolution

This issue was acknowledged by the project team with the comment:
Noted. This locking could only be done via malicious internal action. This can be addressed in a future version of the protocol, and will be avoided through careful operational awareness.

MEHEN-206 - Potential abuse of BurnBridges

Category	Commit	Severity	Status
None	fe63eae1cab7f42d1ebb81b9908df827240bc688	Minor	Resolved

Description

The “Burn Bridges” feature from a previous finding recommendation has been misinterpreted. Lets review why this feature is needed:

Consider the case when the **Owner** keys get compromised.

As soon as the **Owner** keys are compromised, we should begin planning to migrate to a new policy ID; This would likely involve deploying a new protocol that supported a 1-for-1 redemption of the previous token, and a long-term sunsetting plan for the original token.

During that long migration path, the **Owner** key enables an attacker to trigger an upgrade request, which pays the relevant NFTs to a dangerous location.

To prevent this, we allow upgrade requests to be cancelled, delaying the malicious exfiltration of the relevant NFTs.

However, during the long sunsetting period, the actual **Owner** needs to remain ever vigilant against the attacker sneaking in a malicious upgrade request.

For that purpose, we allow the protocol to burn it’s ability to upgrade permanently. This “burns in” the current protocol logic.

As is currently implemented in the contract, the **Burned Bridge** feature also disables minting and burning. It would be highly disruptive to the utility of the **Mehen Asset** if minting was disabled completely during this sunsetting period (indeed, it may be months before a new contract can be written, the community polled, etc.).

It’s worth observing that the protocol is remarkably still safe under the compromise of any one role:

- If the owner is compromised, the attacker can only trigger malicious upgrades subject to a time delay
- If the minting key is compromised, the attacker can only mint USDM up to the limit of the oracle, and only to the distributor address, not their own
- If the distributor key is compromised, the attacker cannot mint any USDM, and can exfiltrate at most a single round of minting.

So the protocol is quite safe even under the compromise of an individual key, so disabling minting completely is too extreme of a step.
Also, if the protocol truly does need to be bricked, the oracle provider can simply stop providing updates regarding the bank balance.

Recommendation

Remove the `!is_locked` condition from the `and` condition in both the **Mehen v1 Control Validator** and the **Mehen v1 Count Validator** . This will allow the protocol to continue minting and burning, while permanently preventing any future protocol changes.
You could also clean up the code slightly after doing so:

- Delete the calculation of the `is_locked` local variable
- Let the `when action` case match be the final condition of the method; i.e. there is no need to store the boolean condition in a local variable, only to make it the single condition of an and block.

Resolution

This issue was resolved as of commit `fe63eae1cab7f42d1ebb81b9908df827240bc688`.

MEHEN-300 - Missing cosmetic guards on protocol boot

Category	Commit	Severity	Status
None		Info	Acknowledged

Description

There are several missing conditions when minting the counter and control NFTs. These don't compromise in any way the security of the protocol, provided these conditions were actually met when the protocol instance is booted. However, they would provide an ultimately cosmetic guard against certain types of user error.

- counter_name and control_nft_name must be distinct, but the script doesn't enforce this
- No other tokens should be minted as part of this transaction, but the script doesn't enforce this
- The tokens should be paid into their respective contracts, with well formed datums, but the script doesn't enforce this

In particular, for example, because the **Mehen v1 Minting Policy** only requires that the **Counter NFT** be present in the outputs, it would allow someone to pre-mint a supply of the token before the protocol was initialized.

Recommendation

We do not believe these need to be addressed, but if you wanted to, you could:

- Construct the expected MultiAsset explicitly, and compare to the transaction mint field
- Enforce that the names in the redeemer are distinct
- Enforce that each NFT is paid to a separate smart contract with the correct datum

Resolution

This issue was acknowledged by the project team with the comment:

Noted. This can be addressed in a future version of the protocol, and will be avoided through careful operational awareness.

MEHEN-301 - Misuse of sundae/multisig

Category	Commit	Severity	Status
None		Info	Acknowledged

Description

In `evaluate_start_timer`, the `owner` **Multisig Script** is checked, using the implementation in the open source Sundae Labs `aicone` library. However, for the dictionary of transaction withdrawals, instead of passing `transaction.withdrawals`, the script passes `dict.new()`.
This means that if `owner` is ever set to a multisig with a `Script` condition, the protocol will be permanently locked.

Recommendation

If you don't plan to use this feature in Mehen v1, no action is neccesary.
Otherwise, you should pass `transaction.withdrawals` for the last parameter of `multisig.satisfied`.

Resolution

This issue was acknowledged by the project team with the comment:
Noted. This can be addressed in a future version of the protocol, and will be avoided through careful operational awareness.

MEHEN-302 - Duplicated files could lead to deployment confusion

Category	Commit	Severity	Status
None		Info	Acknowledged

Description

The audited code paths are the relevant aiken files in the `deno` directory.
However, there are **almost but not quite** duplicate implementations of each script in `lambda/src`
This could lead to confusion when deploying the scripts, and accidentally result in deploying the wrong scripts.

Recommendation

We've attached the manifest of appropriate script hashes and policy IDs at the beginning of this document to avoid confusion.
Still, we recommend deleting these files to avoid confusion, as there are many subtle bugs that went unfixed in this parallel version.

Resolution

This issue was acknowledged by the project team with the comment: We are aware that the final version of the contracts are under the deno repository.

MEHEN-303 - Potential guards through mint diffusion rate

Category	Commit	Severity	Status
None		Info	Acknowledged

Description

One potential extra layer of security and confident that the protocol could implement is a “diffusion rate” for newly minted USDM.
In particular, imagine the scenario where an oracle value gets updated to a high value by mistake, and then shortly thereafter corrected. This could happen from an API error in reporting the bank balance to the oracle, a bug in the oracle, etc.
This would create a brief window in which a large amount of USDM could be minted, despite not being backed by these assets.

Recommendation

While we don’t recommend making any changes now, one future improvement you could add to the protocol is a small time-lock dependent on the size of the minted USDM. For anomalous values, you could require that the reported oracle value **still report** the high value after a time delay, to ensure that it wasn’t an accidental fluke. The size of this time delay could be dependent on the size of the deposit. For example:

- Anything below 1m can be minted immediately
- Anything between 1m and 10m requires a 6 hour time delay
- Anything above 10m requires a 24 hour time delay

This would increase confidence in the system, pushing the boundaries of the trust placed in the system even further.

Resolution

This issue was acknowledged by the project team with the comment: We will discuss for future implementations of the protocol.

MEHEN-304 - Considerations for CIP-68 tokens

Category	Commit	Severity	Status
None		Info	Acknowledged

Description

At the last moment, it was decided to make the USDM token a CIP-68 asset.
There are a number of social engineering attacks to understand with regards to the metadata attached to the reference token that the team should be aware of:

- An attacker could change the decimals places for the token; They could, for example, change it to 0 decimal places, and then send that to someone to pay an invoice and receive some good or service for 1-millionth of the price they should have paid.
  - Smart contracts should be immune to this, as they operate on the diminutive unit without regard to decimal places; the vulnerability here is mostly social.
- An attacker could change the token name to imitate a more valuable token.
- An attacker could attach offensive, derrogatory, illegal, or otherwise inappropriate logos and images to the USDM token.

Recommendation

We recommend treating the CIP-68 reference token with the same level of care os the **Owner** keys. In fact, we recommend paying it to a multisig native script with the same parties as the **Owner** keys.
We recommend native scripts over a new aiken script, as they are simpler, cheaper, and don't risk the deadlocking of funds if no datum is attached.

Resolution

This issue was acknowledged by the project team with the comment: Mehen is going to spend the native script reference token to a multisig native script with the same multisig conditions as the Owner object in the Mehen protocol.

4 - Appendix

4.a - Disclaimer

This Smart Contract Security Audit Report ("Report") is provided on an "as is" basis, for informational purposes only, and should not be construed as investment advice or any other kind of advice on legal, financial, or other matters. The entities and individuals involved in preparing this Report ("Auditors") do not guarantee the accuracy, completeness, or usefulness of the information provided herein and shall not be held liable for any contents, errors, omissions, or inaccuracies in this Report or for any actions taken in reliance thereon.

The Auditors make no claims, promises, or guarantees about the absolute security of the smart contracts audited and the underlying code. The findings, interpretations, and conclusions presented in this Report are based on the best efforts of the Auditors and reflect their professional judgment at the time of the audit. The blockchain and cryptocurrency landscape is rapidly evolving, and new vulnerabilities may emerge that were not identified or considered at the time of the audit. As such, this Report should not be considered as a comprehensive guarantee of the audited smart contracts' security.

The Auditors disclaim, to the fullest extent permitted by law, any and all warranties, whether express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement. The Auditors shall not be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this Report, even if advised of the possibility of such damage.

This Report is not exhaustive and is subject to change without notice. The Auditors reserve the right to update, modify, or revise this Report based on new information, subsequent developments, or further analysis. The Auditors encourage all interested parties to conduct their own independent research and due diligence when evaluating the security of smart contracts.

By using or relying on this Report, you agree to indemnify and hold harmless the Auditors from any claim, demand, action, damage, loss, cost, or expense, including attorney fees, arising out of or relating to your use of or reliance on this Report.

If you have any questions or require further clarification regarding this Report, please contact the contact@sundaeswap.finance.

4.b - Issue Guide

4.b.i - Severity

Severity	Description
Critical	Critical issues highlight exploits, bugs, loss of funds, or other vulnerabilities that prevent the dApp from working as intended. These issues have no workaround.
Major	Major issues highlight exploits, bugs, or other vulnerabilities that cause unexpected transaction failures or may be used to trick general users of the dApp. dApps with Major issues may still be functional.
Minor	Minor issues highlight edge cases where a user can purposefully use the dApp in a non-incentivized way and often lead to a disadvantage for the user.
Info	Info are not issues. These are just pieces of information that are beneficial to the dApp creator, or should be kept in mind for the off-chain code or end user. These are not necessarily acted on or have a resolution, they are logged for the completeness of the audit.
Witness	Witness findings are affirmative findings, which covers bizarre corner cases we considered and found to be safe. Not all such cases are covered, but when something is considered interesting, or might be a common question, we try to include it.

4.b.ii - Status

Status	Description
Resolved	Issues that have been **fixed** by the **project** team.
Mitigated	Issues that have a **partial mitigation**, and are now vulnerable in only **extreme** corner cases.

Acknowledged	Issues that have been **acknowledged** or **partially fixed** by the **project** team. Projects can decide to not **fix** issues for whatever reason.
Identified	Issues that have been **identified** by the **audit** team. These are waiting for a response from the **project** team.

4.c - Revisions

This report was created using a git based workflow. All changes are tracked in a github repo and the report is produced using typst. The report source is available here. All versions with downloadable PDFs can be found on the releases page.

4.d - About Us

Sundae Labs stands at the forefront of innovation within the Cardano ecosystem, distinguished by its pioneering development of the first Automated Market Maker (AMM) Decentralized Exchange (DEX) on Cardano. As a trusted leader in blockchain technology, we offer a comprehensive suite of products and services designed to enhance the Cardano network's functionality and security. Our offerings include Sundae Rewards, Sundae Governance, Sundae Exchange, and Sundae Taste Test—an automated price discovery platform—all available on a Software as a Service (SaaS) basis. These solutions empower other high-profile projects within the ecosystem by providing them with turnkey rewards and governance capabilities, thereby fostering a more robust and scalable blockchain infrastructure.
Beyond our product offerings, Sundae Labs is deeply committed to the advancement of the Cardano community and its underlying technology. We contribute significantly to research and development efforts aimed at improving Cardano's security and scalability. Our engagement with Input Output Global (IOG) initiatives, such as Voltaire, and participation in core technological discussions underscore our dedication to the Cardano ecosystem's growth. Additionally, our expertise extends to software development consulting services, including product design and development, and conducting security audits. Sundae Labs is not just a contributor but a vital partner in Cardano's journey towards achieving its full potential.

4.d.i - Links

Mehen Stablecoin - https://mehen.io
Sundae Labs - https://sundae.fi
Sundae Public Audits - https://github.com/SundaeSwap-finance/sundae-audits-public