Overview of “Mantra” Oracle

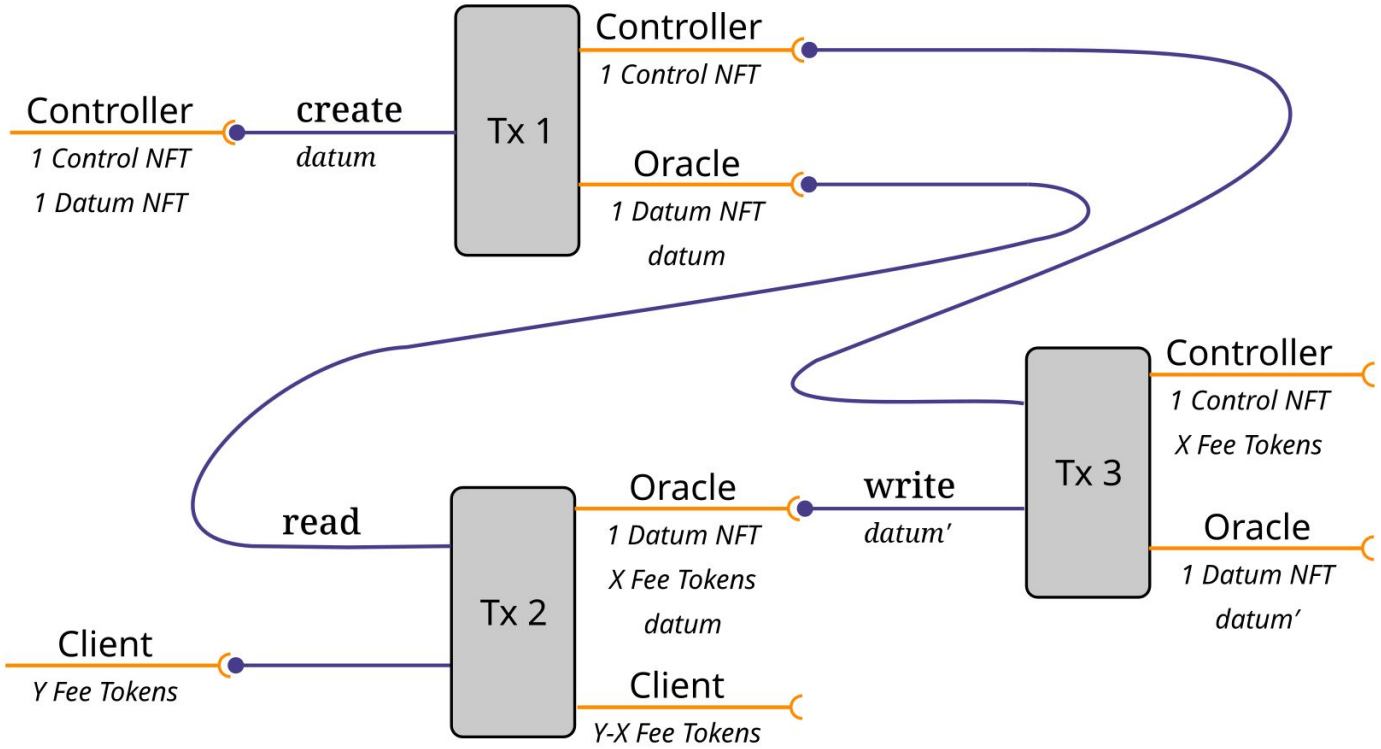
Brian Bush

2 November 2021

A General-Purpose Token-Based Oracle for Cardano

Features
- Easy to operate.
- Configurable fee in ADA and/or token.
- Configurable oracle content (data feeds).
- Suitable for ad-hoc decentralization.
- Secure.

Novelty
- Utilizes a control token.
- Fees paid in any Value.
- Resistant to “nuisance-token” attacks.
- Simultaneous posting of metadata (for user convenience).



Parameterization

The oracle fee may be ADA, a token, a combination, or even negative.

```
-- | Parameters defining the oracle.
data Parameters =
  Parameters
  {
    controlParameter :: AssetClass -- ^ The token needed for writing or deleting the oracle.
  , datumParameter   :: AssetClass -- ^ The token with which the oracle datum is always associated.
  , feeToken         :: AssetClass -- ^ The token in which the fee must be paid for reading the oracle.
  , feeAmount        :: Integer    -- ^ The amount of the fee token needed for reading the oracle, if any.
  , lovelaceAmount   :: Integer    -- ^ The amount of Lovelace needed for reading the oracle, if any.
  }
    deriving (Haskell.Eq, Generic, FromJSON, Haskell.Show, ToJSON)


-- | An oracle controlled by one token, holding another token, and requiring a fee for its use.
data Oracle =
  Oracle
  {
    controlToken :: !AssetClass -- ^ The token needed for writing or deleting the oracle.
  , datumToken   :: !AssetClass -- ^ The token with which the oracle datum is always associated.
  , requiredFee  :: !Value      -- ^ The minimum fee required to read the oracle on-chain.
  }
    deriving (Haskell.Eq, Generic, FromJSON, Haskell.Show, ToJSON)
```

Datum and Redeemer

The datum is any `BuiltinData`, so a wide variety of JSON-like and custom structures may be used.

There are three straightforward redeemers:

```
-- | Redeemers for the oracle.
data Action =
    Delete        -- ^ Delete (close) the oracle.
  | Read          -- ^ Read the oracle's datum.
  | Write         -- ^ Set or update the oracle's datum.
    deriving Haskell.Show

instance Enum Action where
 . . .
```

Validator

```
-- | Make the validator for the oracle.
makeValidator :: Oracle         -- ^ The oracle.
              -> BuiltinData    -- ^ The datum.
              -> Action         -- ^ The redeemer.
              -> ScriptContext  -- ^ The context.
              -> Bool           -- ^ Whether the transaction is valid.
makeValidator Oracle{..} _ redeemer context@ScriptContext{..} =

  let

    -- Oracle input and output.
    continuingOutputs = getContinuingOutputs context
    oracleInput =
      case findOwnInput context of
        Just input -> txInInfoResolved input
        _          -> error ()
    oracleOutput =
      case continuingOutputs of
        [output] -> output
        _        -> error ()

    -- Values.
    valueBefore = txOutValue oracleInput
    valueAfter  = txOutValue oracleOutput
    valueInput  = valueSpent scriptContextTxInfo

   -- Datum token.
    datumFromOracle = assetClassValueOf valueBefore datumToken == 1
    datumToOracle   = assetClassValueOf valueAfter  datumToken == 1

    -- Datum value.
    inputDatumHash  = txOutDatumHash oracleInput
    outputDatumHash = txOutDatumHash oracleOutput
    unchangedDatum = outputDatumHash == inputDatumHash

    -- Control token.
    controlTokenInput = assetClassValueOf valueInput controlToken
    controlToOracle   = assetClassValueOf valueAfter controlToken
    authorizedByControl = controlTokenInput > 0
    noControlToOracle   = deleting || controlToOracle == 0

    -- Deletion.
    deleting = null continuingOutputs

    -- Fee amount.
    feePaid = valueAfter == valueBefore <> requiredFee

  in

    datumFromOracle
      && noControlToOracle
      && case redeemer of
           Delete -> deleting      && authorizedByControl
           Write  -> datumToOracle && authorizedByControl
           Read   -> datumToOracle && unchangedDatum && feePaid
```

Example Validator for Reading the Oracle

```
-- | Make the validator for the reader. This validator
-- | looks up its own key in the oracle daum and then
-- | compares that value to its own redeemer.
makeValidator :: AssetClass        -- ^ The asset class for the datum token.
              -> BuiltinByteString -- ^ The datum.
              -> BuiltinByteString -- ^ The redeemer.
              -> ScriptContext     -- ^ The context.
              -> Bool              -- ^ Whether the transaction is valid.
makeValidator datumToken key expectedValue ScriptContext{..} =
  fromMaybe False
    $ do
      datum <- findOracleValue datumToken scriptContextTxInfo
      object <- fromBuiltinData datum
      actualValue <- lookup key object
      return
        $ actualValue == expectedValue
```

```
-- | Find the oracle value for a transaction.
findOracleValue :: FromData a
                => AssetClass -- ^ The asset class for the datum token.
                -> TxInfo     -- ^ The transaction information.
                -> Maybe a    -- ^ The oracle value, if any.
findOracleValue token txInfo@TxInfo{..} =
  do
    let
      candidates =
        [
          candidate
        | input <- txInfoInputs
        , let candidate = txInInfoResolved input
        , assetClassValueOf (txOutValue candidate) token == 1
        ]
    TxOut{..} <-
      case candidates of
        [candidate] -> Just candidate
        _           -> Nothing
    hash <- txOutDatumHash
    Datum datum <- findDatum hash txInfo
    fromBuiltinData datum
```

Command-Line Interface

```
Usage: mantra-oracle [--version] COMMAND
  Utilities for a Cardano oracle.

Available options:
  -h,--help                Show this help text
  --version                Show version.

Available commands:
  export                   Export the validator and compute its address.
  create                   Create the oracle.
  delete                   Delete the oracle.
  write                    Write a value to the oracle.
  loop                     Update the oracle's value periodically.
  reader                   Export an example validator for reading the oracle
                           and compute its address.
```

Test Cases

- Creating
  - Missing tokens.
  - Incorrect signing key.
  - Successfully create.
- Reading
  - Altering data.
  - Discarding data.
  - Stealing datum token.
  - Stealing everything.
  - No fee.
  - Insufficient fee.
  - Excess fee.
  - Insufficient ADA.
  - Excess ADA.
  - Insufficient fee, excess ADA.
  - Insufficient ADA, excess fee.
  - Successfully reading.
- Writing
  - No control.
  - Stealing datum token.
  - Stealing everything.
  - Depositing control token.
  - Successfully writing.
- Other
  - Illegal redeemer.
  - Another illegal redeemer.
- Deleting
  - No control.
  - Fee instead of control.
  - ADA sent to script.
  - Datum token sent to script.
  - Control token sent to script.
  - Control and datum tokens sent to script.
  - Successfully deleting.

Running daily on mainnet, where it was the fifth Plutus transaction and the first oracle. Also ran on purple and testnet.


```
{
  "disclaimer": "ipfs://QmccBPKZqh9BJTJpC8oM6rc4gBrpcVXqcixX9KCsE6yDKd",
  "oracle": "https://oracle.pigytoken.com",
  "timestamp": "2021-09-18T18:26:21+00:00",
  "data": {
    "nyfed": {
      "source": "https://www.newyorkfed.org",
      "symbols": {
        "SOFR": { "date": "2021-09-16", "value": 5, "scale": 100, "unit": "%",
"url": "https://markets.newyorkfed.org/api/rates/secured/sofr" }
      }
    },
    "coingecko": {
      "source": [
        "Data provided by CoinGecko",
        "https://www.coingecko.com/api"
      ],
      "symbols": {
        "ADAUSD": { "value":       238, "scale":       100, "unit": "USD/ADA" },
        "ADAEUR": { "value":       203, "scale":       100, "unit": "EUR/ADA" },
        "ADAGBP": { "value":       173, "scale":       100, "unit": "GBP/ADA" },
        "ADAIDR": { "value":     33894, "scale":         1, "unit": "IDR/ADA" },
        "ADAJPY": { "value":     26123, "scale":       100, "unit": "JPY/ADA" },
        "ADABTC": { "value":      4905, "scale": 100000000, "unit": "BTC/ADA" },
        "ADAETH": { "value":     68771, "scale": 100000000, "unit": "ETH/ADA" },
        "BTCUSD": { "value":     48435, "scale":         1, "unit": "USD/BTC" },
        "BTCEUR": { "value":     41306, "scale":         1, "unit": "EUR/BTC" },
        "BTCGBP": { "value":     35250, "scale":         1, "unit": "GBP/BTC" },
        "BTCIDR": { "value": 690840384, "scale":         1, "unit": "IDR/BTC" },
        "BTCJPY": { "value":   5324373, "scale":         1, "unit": "JPY/BTC" },
        "BTCETH": { "value":  14026648, "scale":   1000000, "unit": "ETH/BTC" },
        "ETHUSD": { "value":    345359, "scale":       100, "unit": "USD/ETH" },
        "ETHEUR": { "value":    294527, "scale":       100, "unit": "EUR/ETH" },
        "ETHGBP": { "value":    251344, "scale":       100, "unit": "GBP/ETH" },
        "ETHIDR": { "value":  49259826, "scale":         1, "unit": "IDR/ETH" },
        "ETHJPY": { "value":    379650, "scale":         1, "unit": "JPY/ETH" },
        "ETHBTC": { "value":     71327, "scale":   1000000, "unit": "BTC/ETH" }
      }
    },
    "metalslive": {
      "source": "https://api.metals.live",
      "symbols": {
        "Au": { "value":  176020, "scale": 100, "unit": "USD/oz", "date": "2021-09-17T20:59:33Z" },
        "Pt": { "value":   95310, "scale": 100, "unit": "USD/oz", "date": "2021-09-17T20:59:33Z" },
        "Ag": { "value":    2249, "scale": 100, "unit": "USD/oz", "date": "2021-09-17T20:59:33Z" },
        "Pd": { "value":  201793, "scale": 100, "unit": "USD/oz", "date": "2021-09-17T20:59:33Z" },
        "Ir": { "value":  610000, "scale": 100, "unit": "USD/oz", "date": "2021-09-17T20:58:19Z" },
        "Ru": { "value":   75000, "scale": 100, "unit": "USD/oz", "date": "2021-09-17T20:58:19Z" },
        "Rh": { "value": 2025000, "scale": 100, "unit": "USD/oz", "date": "2021-09-17T20:58:19Z" }
      }
    }
  }
}
```

Lessons Learned

- Typed validators are unnecessarily expensive and there are opportunities for tediously crafting low-cost equivalents.
- Oracle cost can be reduced by moving safety (not security) from validators to endpoints.
- Token-based control for oracles improves security, simplifying key rotation and enabling automated response to breaches.
- There are few currency and commodity data feeds (even nominally public ones) whose licensing allows reposting on a blockchain.
    - There is massive non-compliance with data licenses on the web.
    - Coingecko is one of the truly free and repostable sources for cryptocurrency prices.
- It would be straightforward to construct decentralized, loosely federated oracles with cost vs trust vs diversity profiles that are tunable on a user-by-user basis.
    - Value-at-risk computations can drive the optimal “tuning”.
- A Bayesian framework can be applied to track oracle reliability and/or data reliability.

Next Steps

- Reduce script size by manually constructing a custom deserialization of `ScriptContext`.
- Redundancy via an additional CLI command that will watch for the oracle's transactions and then submit a "backup" transaction if the expected transaction was not seen in the specified time window.
- DSL for constructing transactions.
- On-demand execution of oracle.
- Decentralization of the oracle via ad-hoc aggregation.
- Integration with Plutus Application Backend (PAB):
  - Already integrated with "fake" PAB.
  - Awaiting release candidate for PAB.
- Sample contract on mainnet that incentivizes the use of the oracle.
- Educational materials and documentation, mostly aimed at stakepool and other interested operators.

Resources

- Oracle
  - Source
    https://github.com/functionally/mantis-oracle/blob/main/ReadMe.md
  - Haddock documentation
    https://functionally.github.io/mantis-oracle/
  - Live on mainnet
    https://github.com/pigytoken/pigy-delegation/blob/main/oracle/ReadMe.md
  - MIT license
- Supporting Haskell library
  - *Features:* monad transformer stack, minting tokens/NFTs, live/replay watching addresses/coins/scripts on blockchain, etc.
  - Source
    https://github.com/functionally/mantis/blob/main/ReadMe.md
  - Haddock documentation
    https://functionally.github.io/mantis/
  - MIT license