MVP Oracles Designs | Milestone 2

Prepared for: Catalyst Milestone 2 Requirements
Prepared by: Shishir Pai and Sam Lambert

1. Executive Summary

Real-world asset (RWA) applications demand **verifiable**, **multi-point data** to support financial, compliance, and operational use cases. However, most existing oracle architectures were designed for DeFi scenarios, focusing on single-point data feeds like asset prices. These systems fall short for RWA contexts, where complex datasets such as supply chain events, transaction records, or IoT sensor data must be published and verified efficiently.

To address this, we propose a **Merkle Oracle** design that decouples data storage from on-chain validation. Structured around a Merkle Trie and supported by off-chain storage (e.g., IPFS or Cardano-native solutions like Iagon), this design enables selective, trust-minimized publishing of high-volume datasets.

Key features of the design include:

- A **singleton oracle UTxO** with a published Merkle root and associated IPFS CID.
- Off-chain computation of the Merkle Trie from an array of normalized key-value pairs.
- On-chain enforcement of publishing rules, including role-based authorization and auditability.
- Support for future evolution toward decentralized oracle operators and validator-based consensus.

This approach prioritizes **financial data use cases** critical to RWA-DeFi integration, such as:

- Verifying farmer or factory transaction histories to support credit access and cash flow-based lending.
- Enabling programmable finance mechanisms (e.g., insurance claims, supply chain payments) that rely on off-chain events.

While this MVP version is centralized under a trusted oracle operator to streamline early deployment, the system is designed to evolve into a decentralized oracle network with incentives and dispute resolution.

2. Motivation & Limitations with Existing Designs

2.1 Over-Arching Motivation

When dealing with providing data on-chain associated with Real-World Asset (RWAs) use-cases, lots of data points need to be published at once, while reasonably balancing transaction costs.

When doing our research in existing solutions, current flagship oracle designs only support publishing a single data point per oracle group, or at most a small aggregated set. Whilst this has been extremely valuable for the current use-cases we have in DeFi; these designs limit applications that we have as a real-world-asset focused project.

In order to address these limitations; we leverage a **Merkle Oracle** by storing data outside the blockchain (e.g via **IPFS** or Cardano-centric solution such as **Iagon**) and organizing it within a merkle trie, enabling on-chain data verification through proofs.

2.2 Limitations with Existing Designs for RWA Use-Cases

Current oracle designs excel at DeFi applications requiring frequent updates of a singular, highly accurate data point, such as price feeds (e.g price of ADA / USD). A good case-study to highlight is the **Aquarium Protocol** (built by **Fluid**) which leverages the Orcfax cardano oracle data to facilitate automated transactions triggered by price conditions. This is great for sponsored transactions, scheduled payments, or even automated collateralized lending; a great implementation that serves a specific DeFi use-case.

However, **real-world asset** (**RWA**) scenarios introduce additional complexities beyond a single data point. RWAs often require the publication and verification of extensive disjoint datasets on-chain.

Supply-chain management, for instance, demands accurate and verified data across various stages, including financial transactions, harvesting conditions like location and timing, detailed processing information such as certification and batch records, and comprehensive logistics data covering real-time locations, temperature monitoring, and customs documentation.

While initially intended to demonstrate regulatory compliance (such as adherence to import/export standards), this detailed data can also support DeFi uses cases such as some shipping insurance which processes claims triggered by delays or damage, or even issuance of tokenized warehouse receipts which are validated in real time by using IoT sensor data for the reserves.

Current oracle solutions in the market, such as Orcfax, may not have been initially intended to support the high volume of data publications required for RWA protocols. There simply is too much data which needs to be concurrently available and refreshed.

3. Scope and Potential Use-Case of the Merkle Oracle

While not all supply chain data needs to be verified on-chain, some financial interactions are crucial to enable defi to integrate with RWA.

3.1 Use Case #1: Farmer Transaction Data

A key use case is the **recording of transactional data between producers (farmers) and buyers**, such as a farmer selling produce to a factory. In many regions, there is no formal, immutable record of these transactions, farmers have trouble proving their financial records. For example; many payments for such uses-case remain **offline (i.e cash)** or leverage **e-mobile payments**. This **excludes them from traditional credit markets** (i.e without a financial identity or record, they are unable to borrow money to buy inputs such as fertilizers or scale their operations). With our oracle implementation, DeFi lenders can access the data to assess risk and offer capital securely, helping farmers build a verifiable financial identity even in areas where stablecoins or digital payments aren't the default (yet).

3.2 Use Case #2: Factory Transaction Data

Similarly, **factories** seeking financing must often demonstrate their production capacity and sales history. If a honey producer can prove on-chain that it has consistently produced and sold 100 tonnes of honey per year, a lender can use this as a reliable performance indicator. This data provides a stronger foundation than self-reported spreadsheets or static financial reports, which can be easily fabricated or outdated.

3.3 Applications of Financial Records Stored via Oracles for DeFi Protocols

Our framework has the potential to support **two core models of finance**:

- **Collateral-backed lending** – Where warehouse receipts or physical goods (e.g., grain, honey) are tokenized, and their presence is verified through IoT weight sensors or environmental data via the Oracle. If the collateral's weight or condition drops below a threshold, smart contracts can automatically trigger recalls or insurance claims.

- **Cash flow-based lending –** Where borrowers are evaluated based on consistent, verifiable income streams and transaction histories, instead of physical collateral. Oracle-published sales and payment data enables this.

Ultimately, this approach emphasizes selective data capture by storing only a **narrow** but **highly relevant set of financial records**, such as payments from farmer to co-op and co-op to exporter. These records are linked to physical inventory identifiers, such as buckets, drums, or containers of honey, which are stored off-chain but referenced via IPFS and platforms like the Winter Protocol. This structure ensures transparency, facilitates reuse by third parties, and significantly reduces on-chain costs by preventing excessive transaction spam.

4. Case Study: Honey Supply-Chains and Benefits of Leveraging Oracles

4.1 Current Challenges

In regions like Zambia, farmer payments are **typically made in cash** or through **basic mobile money platforms**. While these payments do happen, they **leave no formal digital trace** (for cash transactions). As a result, smallholder producers **lack a financial identity**—there is no immutable **record of income**, **transaction history**, or **repayment behavior**.

Without such records, traditional financial institutions and **lenders cannot accurately assess creditworthiness**, **excluding these farmers from formal credit markets**. In rare cases where borrowing is possible, it is often through exploitative informal lenders with extremely high interest rates. This perpetuates a cycle of exclusion, limiting farmers' ability to invest in inputs, expand production, or withstand climate and market shocks.

4.2 How Oracles Can Help Resolve These Challenges

Through our deployment of **Palmyra Pro** with partners like **Nature's Nectar** in Zambia, we **already collect detailed records on farmer** harvests, **payment transactions**, and field-level activity across more than 3,000 beekeepers. By selectively publishing these financial records on-chain using our **Merkle Oracle**, we can begin building a verifiable financial identity for each farmer.

Key attributes of our approach:

- **No need for behavioral change**: Data already collected through routine processes is repurposed.

- **Privacy-preserving**: Only essential financial data is committed on-chain via hashed, verifiable records.

- **Scalable**: As more farmers are onboarded, the historical depth and breadth of data improves.

This enables a new, tamper-resistant layer of financial trust—anchored in real-world production data—that can be accessed by both traditional and decentralized lenders.

4.3 Intersection of Oracles and DeFi - Lasting Benefits to Farmers Worldwide

Over time, each farmer accrues a transparent, verifiable record of production and payment, establishing a **financial reputation**. With this on-chain footprint:

- **Lenders can assess credit risk** based on real economic activity, not static proxies or unverifiable statements.

- **Farmers gain access to formal credit markets**, reducing reliance on predatory lenders.

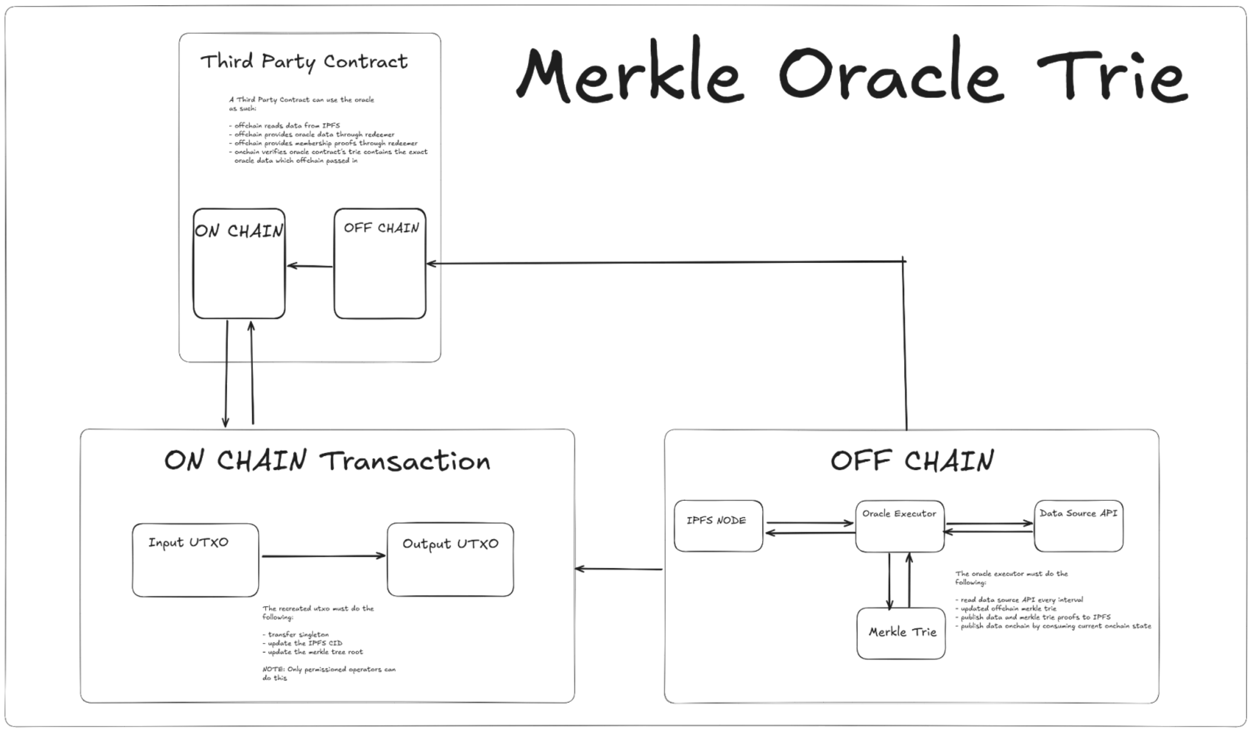
- **Financial inclusion becomes possible** for producers who previously operated outside any system that recognized their contributions.

In short, by **binding payment** and **production data** to an immutable oracle, we convert raw transactions into a digital passport for financial inclusion over time. This infrastructure lays the foundation for integrating rural producers into both **DeFi** and **TradFi** ecosystems—unlocking working capital, improving livelihoods, and fostering sustainable economic growth.

5. System Design and Oracle Architecture

5.1 Overview of Design

The version proposed for this catalyst will be a centralized oracle to serve as a proof of concept for high volume data availability.



A Third Party Contract can use the oracle as such:

- offchain reads data from IPFS
- offchain provides oracle data through redeemer
- offchain provides membership proofs through redeemer
- onchain verifies oracle contract's trie contains the exact oracle data which offchain passed in

The recreated utxo must do the following:

- transfer singleton
- update the IPFS CID
- update the merkle tree root

NOTE: Only permissioned operators can do this

The oracle executor must do the following:

- read data source API every interval
- updated offchain merkle trie
- publish data and merkle trie proofs to IPFS
- publish data onchain by consuming current onchain state

The oracle contract contains the following:

- Singleton Token which identifies the oracle utxo across recreation
- Merkle Trie Root
- IPFS CID

In this **initial** design, there will be only one oracle operator. This operator is trusted. In the future, we may want to extend multiple oracle operators as the system grows and requires such designs to improve trust in the data.

When a datapoint is to be published, the operator will first upload the data as an array of key value pairs. The key is a uuid, while the value must be a primitive value.

```None
[
  {
    "uuid_1": "value1"
  },
  {
    "uuid_2": "value2"
  }
]
```

Then these uuids are inserted into the merkle trie.

If they **already exist in the trie**, it's inserted as an **update operation**.
If **any are removed**, it's **"inserted"** as a **delete operation**.

The data in the merkle trie is to be inserted as such:

```None
trie.insert(blake2b256(uuid_1), blake2b256(value1))
trie.insert(blake2b256(uuid_2), blake2b256(value2))
```

This is done as such as it **normalizes the data to 32 bytes**. The trie serves only to verify rather than store actual data.

NOTE: that the contract does not compute the trie on-chain. All computations are done off-chain, and simply the trie root hash is published on-chain.

The contract will enforce:

- The singleton is sent to the recreated contract output

- Merkle Trie exists
- IPFS CID exists
- Only an authorized address can update the oracle
- Only an authorized address can remove the singleton from the oracle (useful for upgrades)
- `created_at` has a valid timestamp

5.2 Oracle Usage

Offchain can simply read the data by grabbing the latest oracle utxo and sending an http request to fetch the data from IPFS/respective data provider.

Offchain can verify that the IPFS data matches the on-chain merkle trie by querying the IPFS data against the trie. If the key exists and the value matches IPFS, then its known off-chain and on-chain are in sync.

On-chain uses the data which the offchain passes in through the redeemer. On-chain then can verify the value exists in the oracle's trie.

5.3 Building Merkle Trie

The trie can be built by following all the spent oracle utxos in order and inserting merkle trie entries respectively. All the data inserted into the trie is available in IPFS from the datum of the spent oracle utxo. This is a trivial process.

5.4 Design Trade-Offs

This design relies on trust of the oracle operator. This is intentionally done so for this catalyst proposal to serve as an MVP.

It's possible to build on this design by having a validator driven incentived consensus mechanism which would greatly reduce centralization. However, this is non-trivial.

Our MVP design intentionally relies on a trusted oracle operator to simplify initial implementation. By centralizing data publication behind a single operator, we reduce complexity. However, this choice introduces a single point of failure as all trust is on a single operator

Moving to a decentralized model would involve introducing an incentivized validator-driven consensus layer. Nodes would earn reward tokens if the data published is within a certain tolerance with respect to the group.

6. Next Steps and Implementation Requirements

6.1 Cardano Smart Contracts

For the implementation, there will be **two main smart contracts**.

The **first contract** will be a **multisig admin contract** where signatures and threshold details will be stored. The other contract will reference this for any multisig based operations.

The **second contract** will be the **oracle contract** itself. This contract will **store merkle trie root data, and ipfs/data provider reference**. Furthermore, it will make sure only a permissioned set of public keys can spend the contract.

All these operations will be tested with positive and negative cases.

Everything will be documented as well.

6.2 Contract Deployment Suite

This suite will be a command line tool to deploy and run contract operations on both the multisig contract and the oracle contract. It is a way to manually interact with the contracts.

The off-chain code will be tested and documented.

6.3 Off-Chain Requirements

The off-chain implementation will be the most complex of the three. This has to parse data which needs to be published, format it, post to ipfs/data source, maintain merkle trie, and build the on-chain transaction.

The off-chain code will be tested and documented.