CODESPECT

REDSTONE ORACLES

SECURITY ASSESSMENT REPORT

October 6, 2024

Prepared for

RedStone

Contents

1 About CODESPECT

CODESPECT is a specialized smart contract security firm dedicated to ensuring the safety, reliability, and success of blockchain projects. Our services include comprehensive smart contract audits, secure design and architecture consultancy, and smart contract development across leading blockchain platforms such as Ethereum (Solidity), StarkNet (Cairo), and Solana (Rust).

At CODESPECT, we are committed to building secure, resilient blockchain infrastructures. We provide strategic guidance and technical expertise, working closely with our partners from concept development through deployment. Our team consists of blockchain security experts and seasoned engineers who apply the latest auditing and security methodologies to help prevent exploits and vulnerabilities in your smart contracts.

Smart Contract Auditing: Security is at the core of everything we do at CODESPECT. Our auditors conduct thorough security assessments of smart contracts written in Solidity, Cairo, and Rust, ensuring that they function as intended without vulnerabilities. We specialize in providing tailored security solutions for projects on EVM-compatible chains and StarkNet. Our audit process is highly collaborative, keeping clients involved every step of the way to ensure transparency and security. Our team is also dedicated to cutting-edge research, ensuring that we stay ahead of emerging threats.

Secure Design & Architecture Consultancy: At CODESPECT, we believe that secure development begins at the design phase. Our consultancy services offer deep insights into secure smart contract architecture and blockchain system design, helping you to build robust, secure, and scalable decentralized applications. Whether you're working with Ethereum, StarkNet, or other blockchain platforms, our team helps you navigate the complexity of blockchain development with confidence.

Smart Contract Development: In addition to audits, CODESPECT offers smart contract development services. Our team of expert developers specializes in Solidity, Cairo, and Rust, enabling us to build custom, secure, and optimized contracts for your decentralized applications. We work across DeFi, NFT, and other blockchain use cases, always ensuring that security is a priority in every line of code.

With a focus on the intersection of security and innovation, CODESPECT strives to be a trusted partner for blockchain projects at every stage of development.

2 Disclaimer

Limitations of this Audit: This report is based solely on the materials and documentation provided by you to CODESPECT for the specific purpose of conducting the security review outlined in the Summary of Audit and Files. The findings presented in this report may not be comprehensive and may not identify all possible vulnerabilities. CODESPECT provides this review and report on an "as-is" and "as available" basis. You acknowledge that your use of this report, including any associated services, products, protocols, platforms, content, and materials, is entirely at your own risk.

Inherent Risks of Blockchain Technology: Blockchain technology is still evolving and is inherently subject to unknown risks and vulnerabilities. This review focuses exclusively on the smart contract code provided and does not cover the compiler layer, underlying programming language elements beyond the reviewed code, or any other potential security risks that may exist outside of the code itself.

Purpose and Reliance of this Report: This report should not be viewed as an endorsement of any specific project or team, nor does it guarantee the absolute security of the audited smart contracts. Third parties should not rely on this report for any purpose, including making decisions related to investments or purchases.

Liability Disclaimer: To the maximum extent permitted by law, CODESPECT disclaims all liability for the contents of this report and any related services or products that arise from your use of it. This includes but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Third-Party Products and Services: CODESPECT does not warrant, endorse, or assume responsibility for any third-party products or services mentioned in this report, including any open-source or third-party software, code, libraries, materials, or information that may be linked to, referenced by, or accessible through this report. CODESPECT is not responsible for monitoring any transactions between you and third-party providers. We strongly recommend conducting thorough due diligence and exercising caution when engaging with third-party products or services, just as you would for any other product or service transaction.

Further Recommendations: We advise clients to schedule a re-audit after any significant changes to the codebase to ensure ongoing security and reduce the risk of newly introduced vulnerabilities. Additionally, we recommend implementing a bug bounty program to incentivize external developers and security researchers to identify and disclose potential vulnerabilities safely and responsibly.

Disclaimer of Advice: FOR AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, AND ANY ASSOCIATED SERVICES OR MATERIALS SHOULD NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.

3 Risk Classification

Severity Level	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Table 1: Risk Classification Matrix based on Likelihood and Impact

3.1 Impact

- **High** - Results in a substantial loss of assets (more than 10%) within the protocol or causes significant disruption to the majority of users.
- **Medium** - Losses affect less than 10% globally or impact only a portion of users, but are still considered unacceptable.
- **Low** - Losses may be inconvenient but are manageable, typically involving issues like griefing attacks that can be easily resolved or minor inefficiencies such as gas costs.

3.2 Likelihood

- **High** - Very likely to occur, either easy to exploit or difficult but highly incentivized.
- **Medium** - Likely only under certain conditions or moderately incentivized.
- **Low** - Unlikely unless specific conditions are met, or there is little-to-no incentive for exploitation.

3.3 Action Required for Severity Levels

- **Critical** - Must be addressed immediately if already deployed.
- **High** - Must be resolved before deployment (or urgently if already deployed).
- **Medium** - It is recommended to fix.
- **Low** - Can be fixed if desired but is not crucial.

In addition to High, Medium, and Low severity levels, CODESPECT utilizes three other categories for findings: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose a direct security risk but provide useful information the audit team wants to communicate formally.

b) **Best Practices** findings indicate that certain portions of the code deviate from established smart contract development standards.

c) **Undetermined** findings are issues where the impact or likelihood is difficult to predict at the time of the audit.

4 Executive Summary

This document presents the security assessment conducted by CODESPECT for the Redstone oracle protocol's smart contracts. Redstone is an oracle solution that provides customizable and cost-efficient data streams. Unlike traditional oracle providers, Redstone employs a unique mechanism where nodes distribute data to a decentralized data layer. The primary on-chain component of the Redstone oracle verifies the authenticity and reliability of this data, ensuring it originates from trusted nodes.

This audit focuses on two key components of the Redstone oracle: the `Consumer` and `Price Feed` contracts. The consumer part is responsible for data verification, extracting, validating, and aggregating signed data packages. The price feeds use a traditional model where trusted parties update the feed's storage, allowing smart contracts to access stored data. They leverage the consumer's functionality to verify and aggregate data before storing it on-chain, ensuring data reliability and security.

The audit was performed using:

a. manual analysis of the codebase

b. simulation of the smart contract

c. creation of test cases

Along this document, we report 5 points of attention, where one is classified as `Medium`, two classified as `Best Practices` and two classified as `Informational`. The issues are summarized in Table 2.

This document is organized as follows. Section 3 discusses the risk rating methodology. Section 5 summarizes the audit. Section 6 presents the system overview. Section 7 shows found issues. Section 7 contains additional notes for the audit. Section 8 discusses the documentation provided by the client for this audit. Section 9 presents the compilation, tests, and automated tests.

Issues found:

Severity	**Count**	**Fixed**	**Acknowledged**
Critical			
High			
Medium	1	0	0
Low			
Best Practices	2	0	0
Informational	2	0	0
Total	**5**	**0**	**0**

Table 2: Summary of Issues Found, Fixed, and Acknowledged

5 Summary of the Audit

Audit Type	Security Review
Project Name	Redstone
Type of Project	Oracle
Duration of Engagement	2 Weeks
Duration of Fix Review Phase	-
Draft Report	Oct 6, 2024
Final Report	-
Repository	redstone-oracles-monorepo
Commit (Audit)	ff0f3dcb085f28bd80ddc096825701db6e14d0af
Commit (Final)	-
Documentation Assessment	High
Test Suite Assessment	High
Auditors	CODESPECT and Bloqarl

Table 3: Summary of the Audit

5.1 Scope - Audited files

	Contract	LoC	Comments	Ratio	Blank	Total
1	on-chain-relayer/contracts/price-feeds/PriceFeedBase.sol	50	65	130.0%	21	136
2	on-chain-relayer/contracts/price-feeds/MergedPriceFeedAdapterCommon	13	1	7.7%	4	18
3	on-chain-relayer/contracts/price-feeds/PriceFeedsAdapterBase.sol	17	23	135.3%	6	46
4	on-chain-relayer/contracts/price-feeds/without-rounds/MultiFeedAdapterWithoutRounds.sol	177	54	30.5%	47	278
5	on-chain-relayer/contracts/price-feeds/without-rounds/PriceFeedWithoutRounds.sol	15	16	106.7%	5	36
6	on-chain-relayer/contracts/price-feeds/without-rounds/PriceFeedWithoutRoundsForMultiFeedAdapter.sol	27	4	14.8%	7	38
7	on-chain-relayer/contracts/price-feeds/without-rounds/PriceFeedsAdapterWithoutRounds.sol	21	30	142.9%	6	57
8	on-chain-relayer/contracts/price-feeds/without-rounds/MergedPriceFeedAdapterWithoutRounds.sol	41	3	7.3%	9	53
9	on-chain-relayer/contracts/price-feeds/with-rounds/PriceFeedsAdapterWithRounds.sol	82	78	95.1%	24	184
10	on-chain-relayer/contracts/price-feeds/with-rounds/MergedPriceFeedAdapterWithRounds.sol	41	3	7.3%	8	52
11	on-chain-relayer/contracts/price-feeds/with-rounds/PriceFeedWithRounds.sol	31	19	61.3%	12	62
12	on-chain-relayer/contracts/price-feeds/interfaces/IMultiFeedAdapter.sol	10	1	10.0%	8	19
13	on-chain-relayer/contracts/price-feeds/interfaces/IPriceFeedLegacy.sol	5	15	300.0%	4	24
14	on-chain-relayer/contracts/price-feeds/interfaces/IPriceFeed.sol	6	11	183.3%	3	20
15	on-chain-relayer/contracts/core/IRedstoneAdapter.sol	14	60	428.6%	15	89
16	on-chain-relayer/contracts/core/RedstoneAdapterBase.sol	171	169	98.8%	54	394
17	evm-connector/contracts/core/RedstoneConsumerNumericBase.sol	65	63	96.9%	12	140
18	evm-connector/contracts/core/CalldataExtractor.sol	136	35	25.7%	18	189
19	evm-connector/contracts/core/RedstoneConstants.sol	33	15	45.5%	6	54
20	evm-connector/contracts/core/RedstoneDefaultsLib.sol	21	11	52.4%	7	39
21	evm-connector/contracts/core/RedstoneConsumerBase.sol	180	112	62.2%	41	333
22	evm-connector/contracts/libs/SignatureLib.sol	23	2	8.7%	3	28
23	evm-connector/contracts/libs/NumericArrayLib.sol	47	5	10.6%	6	58
24	evm-connector/contracts/libs/BitmapLib.sol	10	3	30.0%	3	16
	Total	**1236**	**798**	**64.6%**	**329**	**2363**

5.2 Summary of Issues

	Issue	Severity	Status
1	Potential Scaling to Unexpected Decimal Places	Medium	Not Fixed
2	Compiler Version with Assembly Bugs	Best Practices	Not Fixed
3	Unused Constant	Best Practices	Not Fixed
4	Potential Rounding Down Could Decrease Overall Price	Info	Not Fixed
5	`MAX_DATA_STALENESS` Should Vary Based on Data Feed	Info	Not Fixed

6 System Overview

Redstone is an oracle protocol that offers customizable and cost-efficient data streams. Unlike traditional oracle providers, **Redstone** uses a unique approach wherein nodes distribute data to a decentralized data layer. These data streams are signed and utilized across various blockchain applications. The main on-chain component of the **Redstone** oracle is responsible for verifying this data, ensuring its reliability and authenticity, and confirming that it originates from trusted nodes.

The CODESPECT audit covers two parts of the Redstone oracle: the Consumer and the Price Feed.

6.1 Consumer

The consumer part contains the core logic of the Redstone oracle. It is responsible for verifying the signed data packages distributed by nodes to the Data Distribution Layer (DDL). These packages are extracted and thoroughly verified to ensure trusted parties sign them. After verification, the aggregated price is returned, which can then be used directly by the smart contract protocol without the need for storage.

The primary workflow of the consumer begins with the inclusion of call data in the transaction, typically provided by the front-end application. This calldata contains the entire data stream fetched from the DDL, which is then extracted and verified on-chain by the consumer contract. The calldata is divided into two parts:

a. **Signed Data**: Consists of individual data packages containing data points with prices.

b. **Unsigned Data:** Contains metadata of the whole stream along with a special marker known as the Redstone marker.

Both parts are crucial for data verification, contributing to the process's overall reliability.

The data is then processed using low-level assembly code, optimizing the procedure for gas efficiency. The contract performs various checks, such as verifying correct data lengths, ensuring the presence of the Redstone marker, and extracting individual data packages along with their timestamps and signatures.

To correctly verify the signed data, each consumer contract must implement the following two functions:

```
function getUniqueSignersThreshold() public view virtual returns (uint8);

function getAuthorisedSignerIndex(address receivedSigner) public view virtual returns (uint8);
```

- The first function returns the number of unique signers. In practice, this indicates how many distinct prices are expected for a specific data feed, with each price signed by a different signer. The data stream generally contains multiple data packages, each signed by a separate signer.
- The second function takes the address of the signer (obtained from the signed message and signature) as input. It checks if the address is a trusted signer. If so, it returns the signer's index; otherwise, the function reverts.

Security Note: To maintain the security and reliability of the data, the consumer contract must enforce a reasonable threshold for unique signers. A higher number of signers leads to greater decentralization and security, though it also increases gas costs for verification. Additionally, each signer must safeguard their private keys to prevent unauthorized data signing.

Once the data has been processed and verified, the final step is to aggregate the prices. By default, the median is used. The median selects the middle price from a sorted array of prices. If the number of elements in the array is even, it returns the arithmetic average of the two middle elements.

6.2 Price Feed

Price feeds follow a more traditional oracle model, where a trusted party updates the feed's storage, allowing all smart contracts to pull data from it. Redstone introduces various types of price feeds, each tailored to meet specific purposes or client requirements. All feeds initially use the consumer's functionality to verify and aggregate signed data from the Data Distribution Layer (DDL). In this case, however, the verified prices are stored on-chain.

The price feed mechanism is divided into two main parts:

- The **price feed contract**, which provides the logic for pulling data from the feed.
- The **price feed adapter**, which generally contains the functionality for updating price data in the feed and validating it.

The core component of the price feed is the `RedstoneAdapterBase` contract. This abstract contract specifies the fundamental functions for updating and retrieving prices, as well as for additional verification:

```
function updateDataFeedsValues(uint256 dataPackagesTimestamp) external;

function getDataFeedIds() external view returns (bytes32[] memory);

function requireAuthorisedUpdater(address updater) external view;
```

- The first function, `updateDataFeedsValues`, is used to update the data of the price feeds. It begins by calling `requireAuthorisedUpdater(...)`, which must be implemented by the specific price feed contract to enforce access control for data updates.
- Next, it checks if the minimum interval between price updates has passed, ensuring that updates do not occur too frequently.
- The `dataPackagesTimestamp` parameter is then validated to ensure it is newer than the oldest stored timestamp and falls within a reasonable update window (by default, it must be no more than 3 minutes in the past and 1 minute in the future).
- After passing all timestamp checks, the contract retrieves an array of data feed IDs for which the feed is responsible using the `getDataFeedIds()` function. This function must be implemented by the underlying feed contract.
- Finally, the updated prices are extracted using the consumer contract's functionality and stored using the `_validateAndUpdateDataFeedsValues(...)` internal function. This function is implemented by specific price feed adapters to handle data validation and storage.

Price feeds are categorized into two types:

a. **Price feeds with rounds**: These feeds implement a round-based system, allowing data to be stored and retrieved with a specific round ID. This feature makes it possible to access historical data.

b. **Price feeds without rounds**: These feeds only provide the latest price, without the ability to return historical data.

Redstone also provides the `MultiFeedAdapterWithoutRounds` contract, which enables the updating and retrieval of data for multiple feeds individually, offering enhanced flexibility.

7 Issues

7.1 [Medium] Potential Scaling to Unexpected Decimal Places

File(s): MultiFeedAdapterWithoutRounds.sol

Description: The MultiFeedAdapterWithoutRounds contract allows updates and price retrieval for multiple feeds. One way to obtain an asset's price is through the priceOf(...) function, which accepts the asset's address as input:

```
function priceOf(address asset) public view virtual returns (uint256) {
    bytes32 dataFeedId = getDataFeedIdForAsset(asset);
    uint256 latestValue = getValueForDataFeed(dataFeedId);
    return convertDecimals(dataFeedId, latestValue);
}
```

This function retrieves the data feed ID using the asset address, gets the latest price, and then scales it to 10^18 decimals using:

```
function convertDecimals(bytes32 /* dataFeedId */, uint256 valueFromRedstonePayload) public view virtual returns
↪ (uint256) {
    // @audit-issue Missing address(dataFeed).decimals() and proper scaling to reflect correct decimals
    return valueFromRedstonePayload * DEFAULT_DECIMAL_SCALER_LAYERBANK;
}
```

The issue is that the price is always multiplied by 10^10, assuming a default of 10^8 decimals from the PriceFeed. However, the function does not consider the actual decimals of the specific data feed. This could lead to a mismatch where the smart contract receives a price in a different decimal format than expected, potentially causing incorrect calculations.

Recommendation(s): Incorporate the data feed's decimals() function to adjust the scaling based on the returned value dynamically.

Status: Unresolved

Update from the client:

7.2 [Best Practices] Compiler Version with Assembly Bugs

File(s): on-chain-relayer/*.sol

Description: The the on-chain-relayer package rely on compiler version(^0.8.14) known to have bugs affecting assembly code blocks. While your current code does not appear to be impacted, it is strongly recommended to upgrade to the latest Solidity version or at least to ^0.8.15, which addresses the bugs detailed here.

Recommendation(s): Upgrade the Solidity compiler to the latest version to ensure the integrity and security of the contracts.

Status: Unresolved

Update from the client:

7.3 [Best Practices] Unused Constant

File(s): RedstoneConstants.sol

Description: The RedstoneConstants contract defines various constants utilized across the evm-connector package. However, the constant FUNCTION_SIGNATURE_BS is not used in any contract.

Removing unused code is a best practice to maintain code clarity.

Recommendation(s): Consider the removal of the FUNCTION_SIGNATURE_BS constant.

Status: Unresolved

Update from the client:

7.4 [Info] Potential Rounding Down Could Decrease Overall Price

File(s): `NumericArrayLib.sol`

Description: The prices for the feeds are calculated as the median of all values. The median requires a sorted array of elements, taking the middle value. However, when the number of elements is even, the median is calculated as the arithmetic average of the two middle elements:

```
uint256 sum = arr[middleIndex - 1] + arr[middleIndex];
return sum / 2;
```

Since Solidity does not support decimal numbers, some precision may be lost during this operation due to rounding down which could lead to different prices than was expected. This behaviour is likely known to the Redstone protocol, but it's important to note this potential loss in precision to maintain reasonable decimal accuracy for the prices.

By default, Redstone feeds use eight decimals, where the expected precision loss is around 0.0000005%, which is considered negligible.

Recommendation(s): Ensure awareness of this rounding behaviour to maintain appropriate precision in price calculations.

Status: Unresolved

Update from the client:

7.5 [Info] `MAX_DATA_STALENESS` Should Vary Based on Data Feed

File(s): `MultiFeedAdapterWithoutRounds.sol`

Description: The `MAX_DATA_STALENESS` constant defines the maximum duration for which price data is considered valid. Once this period expires, the data is marked as stale, and the oracle will revert rather than return this outdated data. The staleness check is conducted in the following function:

```
function _validateLastUpdateDetailsOnRead(bytes32 /* dataFeedId */, uint256 /* lastDataTimestamp */, uint256
 ↪  lastBlockTimestamp, uint256 lastValue) internal view virtual returns (bool) {
    return lastValue > 0 && lastBlockTimestamp + MAX_DATA_STALENESS > block.timestamp;
}
```

Since different data feeds have varying update frequencies (heartbeats), the staleness period should also be tailored to each data feed type to ensure accurate and reliable data.

Recommendation(s): Implement dynamic staleness periods for different data feeds to accommodate their specific update frequencies.

Status: Unresolved

Update from the client:

8 Additional notes

This section provides supplementary auditor observations regarding the code. These points were not identified as individual issues but served as informative recommendations to enhance the overall quality and maintainability of the codebase.

- The `_securelyExtractOracleValuesAndTimestampFromTxMsg(...)` function could potentially revert earlier if `msg.data` is empty or does not meet the minimum expected data length. Alternatively, this check could be enforced in external functions to ensure data integrity.
- Consider checking the `dataFeedIds` argument in `_securelyExtractOracleValuesAndTimestampFromTxMsg(...)` for duplicates to provide a clearer error message. Currently, the function will revert with an `InsufficientNumberOfUniqueSigners` error if duplicates are present, which might not directly indicate the root cause.
- There is inconsistency in typecasting within the code. In some instances, the `SafeCast` library is used, while in others, direct casting (`int256(var)`) is performed after checking the variable range. To enhance code quality, consider adopting a consistent approach. For reference, `SafeCast` is used here, while direct casting is applied here.

9 Documentation Evaluation

Software documentation encompasses the written or visual materials that outline the functionality, architecture, design, and implementation of software. It offers a comprehensive overview of the system, aiding users, developers, and stakeholders in understanding the software's workings, usage, and maintenance. This documentation can come in various forms, such as user guides, system manuals, technical specifications, requirements documents, design documents, and inline code comments. Effective software documentation is crucial in the development process, facilitating clear communication between developers, testers, users, and other stakeholders. It ensures a unified understanding of the software's features and behaviour among all parties involved. Additionally, it significantly enhances software maintenance by providing a detailed reference and simplifying the tasks of modifying, updating, and sustaining the software over time. Smart contracts can benefit from several types of software documentation, with some of the most commonly used being:

- **Technical whitepaper**: A detailed document outlining the smart contract's design and technical aspects, including its purpose, architecture, components, and their interactions;
- **User manual**: A guide that explains how to interact with the smart contract, featuring step-by-step instructions for various operations and an overview of its features and functionalities;
- **Code documentation**: This documentation provides insights into the codebase of the smart contract, covering functions, variables, and classes, along with explanations of their functionality;
- **API documentation**: This document describes the contract's Application Programming Interface (API), detailing the available methods, parameters, and responses for interacting with the contract;
- **Testing documentation**: A report on how the smart contract was tested, including test cases, results, and any issues identified during the testing process;
- **Audit documentation**: Reports, notes, and other materials from the smart contract security audit are crucial for verifying the contract's security and identifying vulnerabilities.

These types of documentation are vital for the development and maintenance of smart contracts. They ensure the contract is correctly designed, implemented, and thoroughly tested, and serve as an essential reference for developers managing future modifications or upkeep.

Feedback for Redstone's Documentation

The **Redstone** team has thoroughly documented their code. Natspec comments explain all functionality in detail and highlight potential risks that could arise from incorrect implementation. Additionally, **Redstone** has provided high-level documentation of their oracle, which is available here. Furthermore, the **Redstone** team was highly responsive and available to address all concerns and questions raised by the **CODESPECT** team.

10 Test Suite Evaluation

10.1 Tests Output

10.1.1 EVM-connector package (Redstone Consumer)

```
> yarn test

Compiled 5 Solidity files successfully (evm target: london).

DataPackagesWrapper
 Should properly execute
 Should work properly with manual payload

DataServiceWrapper
With passed 'dataServiceId'
 Should properly execute with one valid cache
 Should properly execute with one valid and one invalid cache
 Should properly execute with one valid and one slower cache
 Should get urls from @redstone-finance/protocol if not provided
 Should fail if contract doesnt expose getDataServiceId and dataServiceId is not passed
 Should throw error when multiple invalid caches
 Should work with manual payload with all params passed
With RedstoneDataServiceConsumer contract
 Should work with passed urls
 Should work without passed urls
 Should throw on not supported data-service id
 Should work with dataServiceId passed explicit
 Should work with dataServiceId and urls passed explicit
 Should work with manual payload without passed params

SampleBitmapLib
 Bitmap should be empty in the beginning
 Should correctly set bit: 0
 Should correctly set bit: 1
 Should correctly set bit: 42
 Should correctly set bit: 235
 Should correctly set bit: 255

SampleNumericArrayLib
 Should store array in storage
 Should correctly sort values
 Should correctly pick the median value in an array with an odd length
 Should correctly pick the median value in an array with an even length
 Should store array in storage
 Should store array in storage
 Should correctly sort an empty array
 Should revert trying to pick a median value from an empty array
 Should properly sort 1-elem array
 Should correctly pick median from 1-elem array
 Should properly sort 2-elem array
 Should correctly pick median from 2-elem array
 Should properly sort 100-elem array
 Should correctly pick median from 100-elem array

SampleRedstoneDefaultsLib
 Should properly validate valid timestamps
 Should revert for too old timestamp
 Should revert for timestamp from too long future
 Should properly aggregate an array with 1 value
 Should properly aggregate an array with 3 values
 Should properly aggregate an array with 4 values
 Should properly aggregate an array with values, which include a very big number
 Should properly aggregate an array with values, which include zeros
 Should revert trying to aggregate an empty array
```

```
SampleRedstoneConsumerBytesMockManyDataFeeds
 Should properly execute transaction on RedstoneConsumerBase contract (order: ETH, BTC)
 Should properly execute transaction on RedstoneConsumerBase contract (order: BTC, ETH)
 Should work properly with the greater number of unique signers than required
 Should revert if data feed id not found
 Should revert for too old timestamp
 Should revert for different timestamps
 Should revert for an unauthorised signer
 Should revert for insufficient number of signers
 Should revert for duplicated packages (not enough unique signers)

SampleRedstoneConsumerBytesMockStrings
 Should properly execute transaction on RedstoneConsumerBase contract
 Should pass even if there are redundant packages
 Should revert if values from different signers are different
 Should revert if there are too few signers
 Should revert if there are too few unique signers
 Should revert if there is an unauthorised signer

SampleRedstoneConsumerBytesMock
 Should properly execute transaction on RedstoneConsumerBase contract
 Should properly execute if there are redundant packages
 Should properly execute if there are more unique signers than needed
 Should revert if there are too few signers
 Should revert if there are too few unique signers
 Should revert for unauthorised signer
 Should revert for too old timestamp
 Should revert for different timestamps
 Should revert is data feed id not found

Corrupted payload
 Should work properly with the correct redstone payload
 Should revert for corrupted payload (wrong 1 byte in the beginning)
 Should revert for corrupted payload - incorrect (bigger) data points count in the last data package
 Should revert for corrupted payload - incorrect (smaller) data points count in the last data package
 Should revert for corrupted payload - incorrect (bigger) data points count in the first data package
 Should revert for corrupted payload - incorrect (smaller) data points count in the first data package

DuplicatedDataFeeds
 Should get oracle values for empty array
 Should get oracle values for an array with one symbol
 Should get oracle values for feeds with duplicates
 Should get oracle values for feeds with duplicates (100 times BTC)
 Should get oracle values for feeds with duplicates (1000 times ETH)
 Should get oracle values for feeds with duplicates (1 x ETH, 100 x BTC)
 Should get oracle values for feeds with duplicates (100 x BTC, 1 x ETH)
 Should get oracle values for feeds with duplicates (100 x ETH, 1 x BTC)
 Should get oracle values for feeds with duplicates (100 x ETH, 100 x BTC)

SampleWithEvents
 Test events with contract wrapping

Extract Timestamp
 Should extract timestamp correctly
 Should revert if 2 timestamps are not equal
 Should revert if one of many timestamps is different

Long Inputs
 Should pass long bytes oracle value

PopulateTransactionTest
 Should overwrite populateTransaction

SampleChainableProxyConnector
 Should process oracle value for one asset
 Should process oracle values for 10 assets
 Should process oracle values for 10 assets simultaneously
```

```
SampleRedstoneConsumerNumericMock
 Should properly execute transaction on RedstoneConsumerBase contract (ETH)
 Should properly execute transaction on RedstoneConsumerBase contract (BTC)
 Should work properly with the greater number of unique signers than required
 Should revert if data feed id not found
 Should revert for too old timestamp
 Should revert for different timestamps
 Should revert for an unauthorised signer
 Should revert for insufficient number of signers
 Should revert for duplicated packages (not enough unique signers)

SampleProxyConnector
 Should return correct oracle value for one asset
 Should return correct oracle values for 10 assets
 Should forward msg.value
 Should work properly with long encoded functions
 Should fail with correct message (timestamp invalid)
 Should fail with correct message (different timestamps)
 Should fail with correct message (insufficient number of unique signers)
 Should fail with correct message (signer is not authorised)
 Should fail with correct message (no error message)
 Should fail with correct message (string test message)

SampleStorageProxy
 Should return correct oracle value for one asset using dry run
 Should return correct structure containing oracle value using dry run
 Should return correct oracle values for array of values using dry run
 Should return correct array of structures containing oracle values using dry run
 Should return correct structure of arrays containing oracle values using dry run
 Should return correct oracle value for one asset
 Should return correct oracle values for 10 assets
 Should return correct oracle values for 10 assets simultaneously

SignerOrProviderTest
 Should call static function without signer
 Should revert with non-static function without signer
 Should call non-static function with signer

SampleChainableStorageProxy
 Should process oracle value for one asset
 Should process oracle values for 10 assets
 Should process oracle values for 10 assets simultaneously

SampleSyntheticToken
 Maker balance should be 0
 Should mint

verify prices test
- Should properly extract prices with small data packages
- Should properly extract prices with multi point package

Not Wrapped Contract
 Should revert if contract was not wrapped

SampleKydServiceConsumer
- Address should pass KYD
- Address shouldnt pass KYD
- Should revert if invalid response from one node
- Should revert if one value from node is not equal
- Should revert if two calls to the same node
```

```
SampleRedstoneConsumerNumericMockManyDataFeeds
 Should properly execute transaction on RedstoneConsumerBase contract (order: ETH, BTC)
 Should properly execute transaction on RedstoneConsumerBase contract (order: BTC, ETH)
 Should work properly with manual payload
 Should properly execute transaction with 20 single pacakages (10 for ETH and 10 for BTC)
 Should work properly with the greater number of unique signers than required
 Should revert if data feed id not found
 Should revert for enough data packages but insufficient number of one data feed id
 Should revert for too old timestamp
 Should revert for different timestamps
 Should revert for an unauthorised signer
 Should revert for insufficient number of signers
 Should revert for duplicated packages (not enough unique signers)

Simple Mock Numeric Wrapper
 Should properly execute on contract wrapped using simple numeric mock
 Should properly execute on contract wrapped with smaller value byte size
 Should properly execute on contract wrapped with smaller value byte size in one data point
 Should revert for too few signers
 Should revert for too old timestamp
 Should test getting data with timestamp
146 passing (17s)
7 pending
```

10.1.2 On-chain relayer package (Price Feeds)

```
GasReadsBenchmarks
 benchmark
44235
 benchmark hops

MergedAdapterWithoutRoundsSusdeRateProviderBase
 should allow first update
 should allow second update (with the same value) if more than 12 hours passed
 should revert getRate function before the first update
 should read proper value using getRate function
 should allow second update when new value is deviated less than 2% - lesser
 should allow second update when new value is deviated less than 2% - bigger
 shouldn't allow second update if less than 12 hours passed
 shouldn't allow second update when new value is deviated more than 2% - lesser
 shouldn't allow second update when new value is deviated more than 2% - bigger

EthUsdcRedstoneAdapterForFluidOracle
 should properly get indexes for data feeds
 should revert trying to get index if data feed is not supported
 should revert trying to update by unauthorised updater
 should revert if min interval hasn't passed
 should revert if proposed data package timestamp is same as before
 should revert if proposed data package timestamp is older than before
 should revert if proposed data package timestamp is too old
 should revert if proposed data package timestamp is too new
 should revert if at least one timestamp isn't equal to proposed timestamp
 should revert if redstone payload is not attached
 should revert if a data feed is missed in redstone payload
 should revert for too few signers
 should properly update data feeds one time
 should properly update data feeds with extra data feeds in payload
 should properly update data feeds several times
 should get a single data feed value
 should revert trying to get invalid (zero) data feed value
 should revert trying to get a value for an unsupported data feed
 should revert trying to get several values, if one data feed is not supported
 should revert trying to get several values, if one data feed has invalid (zero) value
upgrades
 should properly initialize
should properly upgrade the contract
 should change data feeds
 should change authorized updaters
EthUsdcRedstoneAdapterForFluidOracle specific
 getExchangeRate return 0, when value not set
 getExchangeRate return set value
 benchmark reads

LayerBankOracleAdapterV1
 Should not update values if at least one feed is missing
 Should properly update values one time
 Should fail trying to update any feed with 0 value
 Should properly update values several times
 Should get values using priceOf
 Should get values using pricesOf
 Should get price of ETH using priceOfETH
 Should fail if the data is stale
 Should revert for getting 0 values
 Should revert for an unknown asset
 Should get underlying price
 Should get underlying prices
 Should fail getting underlying prices for an invalid gToken
 Should properly connect PriceFeedWithRounds
```

```
MentoAdapter
 Should report oracle values <test stdout>
 Should not report oracle values when deviation is too big
 Should properly read redstone values reported to sorted oracles

MockSortedOracles
 Different oracles should properly report their values
 The same oracle should properly update their value
 2 iterations of oracle reports
{"date":"2024-10-02T13:56:39.825Z","args":["deviation 97.67441860465101 is higher than max acceptable deviation 10.
 ↪  Sorted oracles median price: 43. RedStone price:1"],"type":"log","level":2,"tag":"mento-utils"}
{"date":"2024-10-02T13:56:39.831Z","args":["deviation 2225.581395348837 is higher than max acceptable deviation 10.
 ↪  Sorted oracles median price: 43. RedStone price:1000"],"type":"log","level":2,"tag":"mento-utils"}
 Too deviated values should not cause updates

RedstonePrimaryProdERC7412
 should properly get indexes for data feeds
 should revert trying to get index if data feed is not supported
 should revert trying to update by unauthorised updater
 should revert if min interval hasn't passed
 should revert if proposed data package timestamp is same as before
 should revert if proposed data package timestamp is older than before
 should revert if proposed data package timestamp is too old
 should revert if proposed data package timestamp is too new
 should revert if at least one timestamp isn't equal to proposed timestamp
 should revert if redstone payload is not attached
 should revert if a data feed is missed in redstone payload
 should revert for too few signers
 should properly update data feeds one time
 should properly update data feeds with extra data feeds in payload
 should properly update data feeds several times
 should get a single data feed value
 should revert trying to get invalid (zero) data feed value
 should revert trying to get a value for an unsupported data feed
 should revert trying to get several values, if one data feed is not supported
 should revert trying to get several values, if one data feed has invalid (zero) value
Tests for getting price feed details
 should properly get data feed id
 should properly get price feed adapter
 should properly get decimals
 should properly get description
 should properly get version
Tests for getting latest price feed values
 should revert calling latestRoundData if the value is zero
 should revert calling latestAnswer if the value is zero
 should revert calling getRoundData if the value is zero
 should properly get latest round data for 1 update
 should properly get latest round data for 2 updates
 should properly get latest answer
 should properly get latest round id for 1 update
 should properly get latest round id for 2 updates
Tests for contract upgrades
 should initialize properly
should properly upgrade the contract
 should change data feed adapter
 should change data feed id
upgrades
 should properly initialize
should properly upgrade the contract
 should change data feeds
 should change authorized updaters
```

```
Tests for getting price feed / adapter details
 should properly get data feed id
 should properly get aggregator address
Tests for main logic of the Merged Adapter and feed
 Should properly update the value first time
 Should properly update the value second time
 Should upgrade the contract
 Should properly update the value third time
```

```
ERC7412 specific logic
 returns oracleId
 should return same value for getDataFeedId and getDataFeedId
 should fail if price was not submitted yet
 should update data using fulfillOracleQuery
 should return cached data
 should revert on stale data

DeviationLib
 return 0 if precision is 0
 should fail if original value is zero
 calculates deviation with new value bigger
 calculates deviation with new value smaller
 calculates deviation with new less round values
 calculates deviation with bigger precision
 calculates deviation with small number and big precision
 calculates deviation with big number and big precision
 calculates deviation with distinct numbers and big precision

CappedPriceFeed
initiation
 can NOT call snapshot before init
 should fail to set cap parameters with too big value
 should fail to set cap parameters with too small value
 should transfer ownership authorized by current owner
 should FAIL to transfer ownership NOT authorized by current owner
 should FAIL to transfer ownership to 0 address
 should allow to set params to new values
 should FAIL to set cap parameters, from unauthorized address
max cap calculation
 should fail to getMaxRatio before setCapParameters was called
 annualGrowth=1000 timePassedSeconds=31536000 => maxRatio=1100000000000000000
 annualGrowth=1000 timePassedSeconds=315360000 => maxRatio=2000000000000000000
 annualGrowth=10 timePassedSeconds=31536000 => maxRatio=1001000000000000000
 annualGrowth=10 timePassedSeconds=31536000 => maxRatio=1001000000000000000
 annualGrowth=10 timePassedSeconds=432000 => maxRatio=1000013698630136986
 annualGrowth=1 timePassedSeconds=1 => maxRatio=1000000000003170979
 annualGrowth=50000 timePassedSeconds=31536000000 => maxRatio=5001000000000000000000
latestAnswer and latestRoundData capped to max
 should return maxRatio when fundamental ratio is bigger then maxRatio
 should return ratio when maxRatio is smaller then maxRatio
market price deviation
 marketRatio=2000000000000000000 fundamentalRatio=2000000000000000000 => deviation=0
 marketRatio=1000000000000000000 fundamentalRatio=2000000000000000000 => deviation=10000
 marketRatio=2000000000000000000 fundamentalRatio=1000000000000000000 => deviation=-5000
 marketRatio=1000000000000000000 fundamentalRatio=1000100000000000000 => deviation=1
 marketRatio=1 fundamentalRatio=100000000 => deviation=999999990000
 should return false when maxMarketDeviationPercent is crossed
 should return true when maxMarketDeviationPercent is NOT crossed
snapshots
 snapshot should revert when fundamental exceeds getMaxRatio
 snapshot works when fundamental ratio increases
 snapshot works when fundamental ratio decreases
 snapshot reverts when fundamental ratio is 0
 snapshot reverts when it was updated recently
 should update cap parameters
 should get snapshot fundamental ratio, when fundamentalRatio() returns 0
 properly snapshots value bigger then 200 bits => 2 ^ 220
 properly snapshots value bigger then 200 bits => 2 ^ 200

upgrade from contracts without rounds to contracts with rounds
 should updates prices and read using priceFeed
```

```
LayerBankOracleAdapterMock
 should properly get indexes for data feeds
 should revert trying to get index if data feed is not supported
 should revert trying to update by unauthorised updater
 should revert if min interval hasn't passed
 should revert if proposed data package timestamp is same as before
 should revert if proposed data package timestamp is older than before
 should revert if proposed data package timestamp is too old
 should revert if proposed data package timestamp is too new
 should revert if at least one timestamp isn't equal to proposed timestamp
 should revert if redstone payload is not attached
 should revert if a data feed is missed in redstone payload
 should revert for too few signers
 should properly update data feeds one time
 should properly update data feeds with extra data feeds in payload
 should properly update data feeds several times
 should get a single data feed value
 should get several data feed values
 should revert trying to get invalid (zero) data feed value
 should revert trying to get a value for an unsupported data feed
 should revert trying to get several values, if one data feed is not supported
 should revert trying to get several values, if one data feed has invalid (zero) value
upgrades
 should properly initialize
should properly upgrade the contract
 should change data feeds
 should change authorized updaters

MergedPriceFeedAdapterWithRounds
 should properly get indexes for data feeds
 should revert trying to get index if data feed is not supported
 should revert trying to update by unauthorised updater
 should revert if min interval hasn't passed
 should revert if proposed data package timestamp is same as before
 should revert if proposed data package timestamp is older than before
 should revert if proposed data package timestamp is too old
 should revert if proposed data package timestamp is too new
 should revert if at least one timestamp isn't equal to proposed timestamp
 should revert if redstone payload is not attached
 should revert if a data feed is missed in redstone payload
 should revert for too few signers
 should properly update data feeds one time
 should properly update data feeds with extra data feeds in payload
 should properly update data feeds several times
 should get a single data feed value
 should revert trying to get invalid (zero) data feed value
 should revert trying to get a value for an unsupported data feed
 should revert trying to get several values, if one data feed is not supported
 should revert trying to get several values, if one data feed has invalid (zero) value
Tests for getting price feed details
 should properly get data feed id
 should properly get price feed adapter
 should properly get decimals
 should properly get description
 should properly get version
Tests for getting latest price feed values
 should revert calling latestRoundData if the value is zero
 should revert calling latestAnswer if the value is zero
 should revert calling getRoundData if the value is zero
 should properly get latest round data for 1 update
 should properly get latest round data for 2 updates
 should properly get latest answer
 should properly get latest round id for 1 update
 should properly get latest round id for 2 updates
Tests for contract upgrades
 should initialize properly
should properly upgrade the contract
 should change data feed adapter
 should change data feed id
upgrades
 should properly initialize
should properly upgrade the contract
 should change data feeds
 should change authorized updaters
Tests for getting price feed / adapter details
 should properly get data feed id
 should properly get aggregator address
Tests for main logic of the Merged Adapter and feed
 Should properly update the value first time
 Should properly update the value second time
 Should upgrade the contract
 Should properly update the value third time

PriceFeedWithRounds
Tests for getting price feed details
 should properly get data feed id
 should properly get price feed adapter
 should properly get decimals
 should properly get description
 should properly get version
Tests for getting latest price feed values
 should revert calling latestRoundData if the value is zero
 should revert calling latestAnswer if the value is zero
 should revert calling getRoundData if the value is zero
 should properly get latest round data for 1 update
 should properly get latest round data for 2 updates
 should properly get latest answer
 should properly get latest round id for 1 update
 should properly get latest round id for 2 updates
Tests for contract upgrades
 should initialize properly
should properly upgrade the contract
 should change data feed adapter
 should change data feed id
Tests for getting historical price feed values
 should properly get round data for several rounds
 should revert trying to get round data for invalid rounds

PriceFeedsAdapterWithRounds
Common adapter tests
 should properly get indexes for data feeds
 should revert trying to get index if data feed is not supported
 should revert trying to update by unauthorised updater
 should revert if min interval hasn't passed
 should revert if proposed data package timestamp is same as before
 should revert if proposed data package timestamp is older than before
 should revert if proposed data package timestamp is too old
 should revert if proposed data package timestamp is too new
 should revert if at least one timestamp isn't equal to proposed timestamp
 should revert if redstone payload is not attached
 should revert if a data feed is missed in redstone payload
 should revert for too few signers
 should properly update data feeds one time
 should properly update data feeds with extra data feeds in payload
 should properly update data feeds several times
 should get a single data feed value
 should get several data feed values
 should revert trying to get invalid (zero) data feed value
 should revert trying to get a value for an unsupported data feed
 should revert trying to get several values, if one data feed is not supported
 should revert trying to get several values, if one data feed has invalid (zero) value
upgrades
 should properly initialize
should properly upgrade the contract
 should change data feeds
 should change authorized updaters
Tests for adapter with rounds support
 should properly get latest round id
 should properly get latest round params
 should properly get values for different (valid) rounds
 should revert trying to get values for invalid rounds
 should properly get values and timestamps for different (valid) rounds
 should revert trying to get values and timestamps for invalid rounds

MergedPriceFeedAdapterWithoutRounds
 should properly get indexes for data feeds
 should revert trying to get index if data feed is not supported
 should revert trying to update by unauthorised updater
 should revert if min interval hasn't passed
 should revert if proposed data package timestamp is same as before
 should revert if proposed data package timestamp is older than before
 should revert if proposed data package timestamp is too old
 should revert if proposed data package timestamp is too new
 should revert if at least one timestamp isn't equal to proposed timestamp
 should revert if redstone payload is not attached
 should revert if a data feed is missed in redstone payload
 should revert for too few signers
 should properly update data feeds one time
 should properly update data feeds with extra data feeds in payload
 should properly update data feeds several times
 should get a single data feed value
 should revert trying to get invalid (zero) data feed value
 should revert trying to get a value for an unsupported data feed
 should revert trying to get several values, if one data feed is not supported
 should revert trying to get several values, if one data feed has invalid (zero) value
Tests for getting price feed details
 should properly get data feed id
 should properly get price feed adapter
 should properly get decimals
 should properly get description
 should properly get version
Tests for getting latest price feed values
 should revert calling latestRoundData if the value is zero
 should revert calling latestAnswer if the value is zero
 should revert calling getRoundData if the value is zero
 should properly get latest round data for 1 update
 should properly get latest round data for 2 updates
 should properly get latest answer
 should properly get latest round id for 1 update
 should properly get latest round id for 2 updates
Tests for contract upgrades
 should initialize properly
should properly upgrade the contract
 should change data feed adapter
 should change data feed id
upgrades
 should properly initialize
should properly upgrade the contract
 should change data feeds
 should change authorized updaters
Tests for getting price feed / adapter details
 should properly get data feed id
 should properly get aggregator address
Tests for main logic of the Merged Adapter and feed
 Should properly update the value first time
 Should properly update the value second time
 Should upgrade the contract
 Should properly update the value third time
```

```
MultiFeedAdapterWithoutRounds
Common multi price feed adapter tests
 should update 1 value
 should update 2 values
 Should get details of the latest update for many feeds
 should update different subsets of values
 should revert when trying to get details of feed that wasn't updated yet
 should revert if redstone payload is not attached
 should revert for an unauthorised signer
 should revert for too few signers
 should properly update data feeds with extra data feeds in payload
 should revert trying to update with duplicated feeds in an array
 should revert trying to update a missing feed
 should revert trying to get a zero value for a data feed
 should properly get several values
 should revert trying to get several values, if one of values is zero
 should revert if proposed data package timestamp is too old
 should revert if proposed data package timestamp is too new
Test values of different size, up to 32 bytes
 should properly save and retrieve value: 1
 should properly save and retrieve value: 10
 should properly save and retrieve value: 100000
 should properly save and retrieve value: 1231242141
 should properly save and retrieve value: 1231242141123123126767
 should properly save and retrieve value: 251635216352137128372183671267
 should properly save and retrieve value: 251635216352137128372183671261267321763721677
 should properly save and retrieve value: 2516352163521371283721836712612673217637216772321321321312312312312321
 should properly save and retrieve value: 25163521635213712837218367126126732176372167723213213213123123123123212321321
 should properly save and retrieve value: 45792089237316195423570985008687907853269984665640564039457584007913129639930
 should properly update values of different size for the same feed
Tests of the updates independency for data feeds (value validation)
Value validation per feed
 should not fail if all feeds are invalid due to value
 should update all valid feeds skip the rest (some feeds are valid, some - invalid)
 should update all valid feeds skip the rest (only one feed is invalid)
 should update all valid feeds skip the rest (only one feed is valid)
Block timestamp validation per feed
 should not fail if all feeds have been updated in the same block
 should properly skip updates for feeds with the same block.timestamp
Data timestamp validation per feed
 should not fail if all feeds have the same data timestamp as before
 should properly skip updates for feeds with an older data timestamp than before
 should properly skip updates for feeds with the same data timestamp as before
Benchmark gas costs for view functions
Gas for getLastUpdateDetailsUnsafe: 24030
 Gas cost for getLastUpdateDetailsUnsafe
Gas for getLastUpdateDetails: 24276
 Gas cost for getLastUpdateDetails
- Benchmark for reading last update details of up to 200 feeds
Tests for getting price feed details
 should properly get data feed id
 should properly get price feed adapter
 should properly get decimals
 should properly get description
 should properly get version
Tests for getting latest price feed values
 should revert calling latestRoundData if the value is zero
 should revert calling latestAnswer if the value is zero
 should revert calling getRoundData if the value is zero
 should properly get latest round data for 1 update
 should properly get latest round data for 2 updates
 should properly get latest answer
 should properly get latest round id for 1 update
 should properly get latest round id for 2 updates
Tests for contract upgrades
 should initialize properly
should properly upgrade the contract
 should change data feed adapter
 should change data feed id
```

```
PriceFeedWithoutRounds
Tests for getting price feed details
 should properly get data feed id
 should properly get price feed adapter
 should properly get decimals
 should properly get description
 should properly get version
Tests for getting latest price feed values
 should revert calling latestRoundData if the value is zero
 should revert calling latestAnswer if the value is zero
 should revert calling getRoundData if the value is zero
 should properly get latest round data for 1 update
 should properly get latest round data for 2 updates
 should properly get latest answer
 should properly get latest round id for 1 update
 should properly get latest round id for 2 updates
Tests for contract upgrades
 should initialize properly
should properly upgrade the contract
 should change data feed adapter
 should change data feed id
Tests for getting historical price feed values
 should revert trying to get round data for invalid rounds

PriceFeedsAdapterWithRounds
 should properly get indexes for data feeds
 should revert trying to get index if data feed is not supported
 should revert trying to update by unauthorised updater
 should revert if min interval hasn't passed
 should revert if proposed data package timestamp is same as before
 should revert if proposed data package timestamp is older than before
 should revert if proposed data package timestamp is too old
 should revert if proposed data package timestamp is too new
 should revert if at least one timestamp isn't equal to proposed timestamp
 should revert if redstone payload is not attached
 should revert if a data feed is missed in redstone payload
 should revert for too few signers
 should properly update data feeds one time
 should properly update data feeds with extra data feeds in payload
 should properly update data feeds several times
 should get a single data feed value
 should get several data feed values
 should revert trying to get invalid (zero) data feed value
 should revert trying to get a value for an unsupported data feed
 should revert trying to get several values, if one data feed is not supported
 should revert trying to get several values, if one data feed has invalid (zero) value
upgrades
 should properly initialize
should properly upgrade the contract
 should change data feeds
 should change authorized updaters

SinglePriceFeedAdapter
 should properly get indexes for data feeds
 should revert trying to get index if data feed is not supported
 should revert trying to update by unauthorised updater
 should revert if min interval hasn't passed
 should revert if proposed data package timestamp is same as before
 should revert if proposed data package timestamp is older than before
 should revert if proposed data package timestamp is too old
 should revert if proposed data package timestamp is too new
 should revert if at least one timestamp isn't equal to proposed timestamp
 should revert if redstone payload is not attached
 should revert if a data feed is missed in redstone payload
 should revert for too few signers
 should properly update data feeds one time
 should properly update data feeds with extra data feeds in payload
 should properly update data feeds several times
 should get a single data feed value
 should revert trying to get invalid (zero) data feed value
 should revert trying to get a value for an unsupported data feed
 should revert trying to get several values, if one data feed is not supported
 should revert trying to get several values, if one data feed has invalid (zero) value
upgrades
 should properly initialize
should properly upgrade the contract
 should change data feeds
 should change authorized updaters

SinglePriceFeedAdapterWithClearing
 should properly get indexes for data feeds
 should revert trying to get index if data feed is not supported
 should revert trying to update by unauthorised updater
 should revert if min interval hasn't passed
 should revert if proposed data package timestamp is too old
 should revert if proposed data package timestamp is too new
 should revert if at least one timestamp isn't equal to proposed timestamp
 should revert if redstone payload is not attached
 should revert if a data feed is missed in redstone payload
 should revert for too few signers
 should properly update data feeds one time
 should properly update data feeds with extra data feeds in payload
 should properly update data feeds several times
 should get a single data feed value
 should revert trying to get invalid (zero) data feed value
 should revert trying to get a value for an unsupported data feed
 should revert trying to get several values, if one data feed is not supported
 should revert trying to get several values, if one data feed has invalid (zero) value
upgrades
 should properly initialize
should properly upgrade the contract
 should change data feeds
 should change authorized updaters <test stdout>
 should temporary set update interval when time conditions DOES NOT exists <test stdout>
 should update prices in mento adapter

Chain config presence
 There should be a chain config for each chain used in relayer manifests

Price feed contract should return the same dataFeedId as in relayer manifest
 abracadabraKavaBtc
 abracadabraKavaEth
 abracadabraKavaUsdt
 arbitrumAngleAgeur
 arbitrumPremia
 arbitrumSusdeRateProvider
 arbitrumWeetheth
 arbitrumWeethfundamental
 arbitrumXvs
```

```
b2MultiPriceFeed
baseBsdetheth
baseEusd
basePufetheth
baseUsdplus
berachainTestnetEth
blastBtc
blastEth
blastLrts
blastTestnet
blastUsdb
bnbBnb
bnbBtc
bnbEzetheth
bnbStone
bnbWeethfundamental
bobMultiPriceFeed
cadenceCantoAtom
cadenceCantoCanto
cadenceCantoCnote
cadenceCantoEth
cadenceCantoTestnet
cadenceCantoUsdc
cadenceCantoUsdt
cyberEth
ethereumApxetheth
ethereumC3m
ethereumEtherfiWeeth
ethereumEtherfiWeetheth
ethereumEthpluseth
ethereumEthxeth
ethereumEusd
ethereumEzetheth
ethereumPufStaking
ethereumPufetheth
ethereumRsetheth
ethereumRswetheth
ethereumSfrxetheth
ethereumStakewiseOsetheth
ethereumUsdeSusde
etherlinkGhostnetTezosXtzEthBtc
etherlinkTezos
fraxtalPackage
lineaEzetheth
lineaWeethfundamental
mantaLayerBank
mantleEth
mantleMnt
mantleUsdeSusde
mantleUsdt
mantleWstEth
mentoBaklavaMultisig
mentoCeloMainnet
merlinMerl
merlinMultiPriceFeed
modeLayerBank
modeMode
modeUsde
optimismApxetheth
realGbp
realXau
seiMultiAdapter
sepoliaAngleAgeur
sepoliaVenusXvs
staderEthx
swell
venusBnbTestnet
venusBnbTrx
venusMainnetXvs
zksyncZk
arbitrumOneMultiFeed
```

```
 bnbMultiFeed
 ethereumMultiFeed
 haven1TestnetMultiFeed
 hemiTestnetMultiFeed
 mantleMultiFeed
 modeMultiFeed
 optimismDevMultiFeed
 optimismMultiFeed
 scrollMultiFeed
 seiMultiFeed
 sepoliaMultiFeed
 zircuitMultiFeed
 zkLinkMultiFeed
 zksyncMultiFeed
cron-condition
 should properly return false if was recently updated
 should properly return false if was updated some time ago
 should properly return true
 should return true if one of multiple cron expressions fulfilled
 should return true if two of multiple cron expressions fulfilled
 should return false if none of multiple cron expressions fulfilled

fallback-cron-condition
 should properly return false as it would return even without fallback
 should properly return false due to offset
 should return true if time diff bigger than interval increased by offset

fallback-time-condition
 should return false if time diff smaller than interval
 should return false if time diff bigger than interval but less than interval increased by offset
 should return true if time diff bigger than interval increased by offset

should-update
 should return false if all checks fail <test output>
 should return true if value-deviation check succeed
 should return true if time check succeed
 should return true for same value when data packages contains custom decimals
 should return true for smaller value when data packages contains custom decimals

time-condition
 should return false if time diff smaller than interval
 should return true if time diff bigger than interval

value-deviation-condition fallback mode not lazy tests
 should return false if older value diff bigger than expected but latest not
 should return false if latest value diff bigger than expected but older not
 should return true if both latest and older values diff bigger than expected
 should return false if both latest and older values diff lower than expected

value-deviation-condition fallback mode tests
 should return false if older value diff bigger than expected but latest not
 should return false if latest value diff bigger than expected but older not
 should return true if both latest and older values diff bigger than expected
 should return false if not enough time passed since last update
 should return true if skip frequent updates enabled and enough time passed since last update
 should return false if both latest and older values diff lower than expected

check-value-deviation-condition
 should return false if value diff smaller than expected
 should return true if value diff bigger than expected
```

Feedback for Redstone's test suite

The **Redstone** team crafted a detailed test suite that covers various flows simulating real use-cases of the oracles. The tests for data packing and verification are well-designed, aiding in the overall understanding of data processing and showcasing well-prepared test scenarios. Additionally, these tests ensure the robustness and reliability of the oracle's data handling mechanisms.