Introduction

A time-boxed security review of the **DORA** protocol was done by **ddimitrov22** and **chrisdior4** , with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

About **DORA**

DORA means Distributed Oracle Agreement. The protocol aims to bridge the gap between the blockchain and the off-chain by providing reliable data to the blockchain. An important note is that the off-chain functionality of DORA was NOT in the scope of this audit. Here is how the workflow of the protocol:

1. Data sources are mapped to the clan nodes to provide them with information
    - The mapping between the data sources and the nodes is done by a VRF (this mitigates the risk of Byzantine data sources)
2. Clan nodes
    - Once a node has obtained data from various data sources, it computes a median(because it allows us to be resilient against data source corruption of less than 50%). Then the clan nodes send the median to multiple aggregators.
3. Aggregators(here a voting for a coherent cluster is happening)(if this voting fails, a fallback mechanism is activated)
    - Once a coherent cluster is formed, the aggregator compute the arithmetic mean or the average of this values
    - Then the aggregator sends the mean and the cluster to all the clan nodes for approval, which from their side validates if the cluster is indeed coming from the values of the clan nodes. If we have more than half of the votes send to the aggregator, a quorum is reached.
4. Then aggregators send transactions(with a S value) which the SMR protocol orders in a log. The first value that is posted to the SMR chain is the authoritative value coming from the oracle.

These are the core contracts in the protocol:

- DoraConsumer - calculates the S value of the different pairs
- SupraSValueFeedStorage - stores the exchange rate of trading pairs

- SupraSValueFeedVerifier - verifies Oracle SMR Transactions using BLS Signatures and stores the price data

Full documentation here.

Severity classification

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

Impact - the technical, economic and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - 27f26660a87353387cde24d12979186f54e38b50**

Scope

The following smart contracts were in scope of the audit:

- `DoraConsumer`
- `ISupraSValueFeed`
- `SupraSValueFeedStorage`
- `SupraSValueFeedVerifier`

The following number of issues were found, categorized by their severity:

- Critical & High: 1 issue
- Medium: 0 issues
- Low: 2 issues
- Informational: 6 issues

Findings Summary

ID	Title	Severity
[H-01]	All of the pairings' timestamps will be updated for epoch 0	High
[L-01]	A `require` check is placed at the wrong place	Low

ID	Title	Severity
[L-02]	Use `Ownable2Step` instead of `Ownable`	Low
[I-01]	NatSpec docs are incomplete	Informational
[I-02]	Use `onlyOwner` modifier instead of repetitive `require` checks	Informational
[I-03]	Prefer Solidity Custom Errors over `require` statements with strings	Informational
[I-04]	Missing event emissions in state changing methods	Informational
[I-05]	Variables can be turned into an `immutable`	Informational
[I-06]	Change function visibility `public` to `external`	Informational

Detailed Findings

[H-01] All of the pairings' timestamps will be updated for epoch 0

Severity

Impact: High, as the epochs are important part of the protocol and the logic will be broken.

Likelihood: High, as it will happen every time a cluster is processed.

Description

An `epoch` is a pre-determined period of time which is used to decide when to reorganize tribes and redraw the clans. Therefore, it is used to store the values of any pairings for a given `epoch` in a mapping. The variable is initialized with default value 0 inside `SupraSValueFeedVerifier` :

```
uint256 epochCount = 0;
```

Then it is used when the `processCluster` to `getTimestamp` and to update the timestamp by calling `restrictedSetTimestamp` :

```
    function processCluster(Smr.Vote memory vote, Smr.MinBatch memory
smrBatch, Smr.MinTxn memory smrTxn, Smr.SignedCoherentCluster memory scc,
uint batchIdx, uint txnIdx, uint clusterIdx, uint256[2] calldata sig)
public {
        ...

        for (uint i = 0; i < cluster.pair.length; i++) {
            uint pair = cluster.pair[i];
            uint timestamp = cluster.timestamp[i];
            uint prevTimestamp = supraSValueFeedStorage.getTimestamp(pair,
epochCount);
            if (prevTimestamp > timestamp) {
```

```
                continue;
            }
            supraSValueFeedStorage.restrictedSetTimestamp(pair,
epochCount, timestamp);
        }
        ...
    }
```

The problem arises from the fact that every time the calls to these functions are made, a value of 0 will be passed for the `epoch` parameter. The `epochCount` variable is not updated anywhere across the contracts. If we take a look at how `restrictedSetTimestamp` is by performing checks that only the `SupraSValueFeedVerifier` can call it and then calls `setTimestamp` which looks like this:

```
    function setTimestamp(uint _tradingPair, uint256 _epoch, uint
timestamp) internal {
        latestTimestamp[_tradingPair][_epoch] = timestamp;
    }
```

This means that the `timestamp` will be always updated for `epoch` 0 for the specified `_tradingPair`. Thus, the logic of using `epochs` is broken and they are completely unnecessary in the current implementation.

Recommendations

Consider to implement a functionality which updates the `epoch`. This can be done by updating the epoch when a specific number of clusters are processed or when a given amount of time has passed (e.g. 2 days) as described in the documentation.

CLIENT

Acknowledged - corrected.

[L-01] A `require` check is placed at the wrong place

Description

The `restrictedSetTimestamp` function is checking that `supraSValueFeedVerifier` is not `address(0)`.

```
    function restrictedSetTimestamp(uint _tradingPair, uint256 _epoch,
uint timestamp) external {
        require(supraSValueFeedVerifier != address(0),
"SupraSValueFeedStorage: Set address for Verifier contract");
        require(msg.sender == supraSValueFeedVerifier,
```

```
"SupraSValueFeedStorage: Not authorised to update data");
        setTimestamp(_tradingPair, _epoch, timestamp);
    }
```

However, that check is unnecessary because `address(0)` is not able to call any functions as no one has access to it. At the same time, `updateSupraSValueFeedVerifier` is missing this check which means that the `supraSValueFeedVerifier` can actually be set to `address(0)`.

Recommendations

Add the `require` check to `updateSupraSValueFeedVerifier` and remove it from `restrictedSetTimestamp`:

```
    function updateSupraSValueFeedVerifier(address
_supraSValueFeedVerifier) public {
        require(msg.sender == owner(), "SupraSValueFeedStorage:
Unauthorised Access");
+       require(_supraSValueFeedVerifier != address(0), ...);
        supraSValueFeedVerifier = _supraSValueFeedVerifier;
    }
```

CLIENT

Acknowledged - corrected.

[L-02] Use `Ownable2Step` instead of `Ownable`

Consider using the `Ownable2Step` library by `OpenZeppelin` as it is using two-step transfer of the ownership. This ensures that the ownership of the contracts cannot be transferred to a wrong address as it needs to be claimed. Also, the `OpenZeppelin` library is well-tested and optimized.

CLIENT

Acknowledged - corrected.

[I-01] NatSpec docs are incomplete

`DoraConsumer`'s external method `getSupraSValue` is missing NatSpec. Also couple of methods are missing `@return` NatSpec such as: `getSvalue` and `getSvalues`. NatSpec documentation is essential for better understanding of the code by developers and auditors and is strongly recommended. Please refer to the NatSpec format and follow the guidelines outlined there.

CLIENT

Acknowledged - corrected.

[I-02] Use `onlyOwner` modifier instead of repetitive `require` checks

In `updatePublicKey` and `updateSupraSValueFeedVerifier` we have a require statement which checks if the `msg.sender` is the `owner`. Instead delete this check and add the `onlyOwner` modifier to both of the methods.

CLIENT

Acknowledged - corrected.

[I-03] Prefer Solidity Custom Errors over `require` statements with strings

Using Solidity Custom Errors has the benefits of less gas spent in reverted transactions, better interoperability of the protocol as clients of it can catch the errors easily on-chain, as well as you can give descriptive names of the errors without having a bigger bytecode or transaction gas spending, which will result in a better UX as well. Consider replacing the `require` statements with custom errors.

CLIENT

Acknowledged - corrected.

[I-04] Missing event emissions in state changing methods

It's a best practice to emit events on every state changing method for off-chain monitoring. The `updateSupraSValueFeedVerifier`, `restrictedSetTimestamp` , `restrictedSetSupraStorage` and `updatePublicKey` methods are missing event emissions, which should be added.

CLIENT

Acknowledged - corrected.

[I-05] Variables can be turned into an `immutable`

The `supraFeed` state variable in `DoraConsumer` as well as `domain`, `blsPrecompileGasCost` and `supraSValueFeedStorage` variables in `SupraSValueFeedVerifier` can be made `immutable` since they are only set in the constructor and never changed after that.

CLIENT

Acknowledged - corrected.

[I-06] Change function visibility `public` to `external`

If the function is not called internally, it is cheaper to set your function visibility to external instead of public.

- `processCluster`
- `checkPublicKey`

CLIENT

Acknowledged - corrected.