Hi,

I am sharing the implementation of a Binance oracle suapp.

Design

The design is composed of an oracle contract on the SUAVE chain and a settlement contract on Goerli.

- The oracle contract leverages MEVM to:
- query Binance API,
- create and sign a Goerli transaction with the price update,
- submit tx signed tx either via public RPC or as a bundle to Flashbots builder.
- query Binance API,
- create and sign a Goerli transaction with the price update,
- submit tx signed tx either via public RPC or as a bundle to Flashbots builder.
- The settlement contract receives updates coming from a designated address and stores the latest price update.

Implementation

Below is the implementation of an oracle contract.

github.com

halo3mic/suave-playground/blob/18c7e190f25794dcecb40f62ac69688756e19ab1/contracts/oracle/BinanceOracle.sol

// SPDX-License-Identifier: MIT // Author: Miha Lotric (halo3mic)

pragma solidity ^0.8.13;

import { AnyBundleContract, Suave } from "../standard_peekers/bids.sol"; import { SuaveContract } from "../blockad/lib/SuaveContract.sol"; import "../../node_modules/solady/src/utils/JSONParserLib.sol"; import "../libraries/Transactions.sol"; import "../libraries/Bundle.sol"; import "solady/src/utils/LibString.sol";

contract BinanceOracle is SuaveContract { using JSONParserLib for *;

uint public constant GOERLI_CHAINID = 5;
string public constant GOERLI_CHAINID_STR = "0x5";
uint8 public constant DECIMALS = 4;
string public constant S_NAMESPACE = "oracle:v0:pksecret";

This file has been truncated. show original

Notice that the private key is generated, stored within MEVM and is never leaked outside the kettle; but the address associated with it is made public. This address is used on Goerli side to verify the price update is valid and is actually coming from a kettle.

The implementation of Goerli settlement contract can be foundhere.

To run your own implementation check outthe oracle docs detailing how to deploy and run the demo.

Live demonstration

The contracts are live:

- settlement contract: 0x04841bd5f6f1f0bab87a06ea3f584e9f9af18556 (Goerli)
- oracle contract: 0x973c7C45dF5b0386230fBb0F414484cB704EEfEF (SUAVE Chain)

Note that this is merely POC that explores capabilities provided by SUAVE with the Rigil version and it is not intended to promote or discourage oracle design with SGX/SUAVE. I believe a lot more thought is needed to consider the risks and to arrive at appropriate design. I hope this post opens such discussion!