Balancer Shared Pool Price Provider

Summary

This document describes the implementation of a Balancer adapter (Price Provider) to get the price in ether for a shared pool token also known as shared BPT (Shared Balancer Pool Token).

Requirements

The implementation of the Balancer adapter must fulfill this requirements:

- Pool token price cannot be manipulated
- Chainlink will be used as the main oracle
- It should use as less gas as possible
- Limited to Balancer's shared pools where the weights cannot be changed
- Limited to a pool containing 2 to 3 tokens

Price manipulation

There can be two potential attacks that could manipulate the pool token price: trade attack and gulp attack.

Trade attack

Balancer pools are susceptible to token reserves manipulation. For example an attacker can execute a flashloan to make a large trade, shift the balances of the token reserves and increase the pool's total value in ethers. This might result in an inflated price of the pool token.

Fortunately, this attack is already limited by some trading restrictions baked into the Protocol on the maximum swap in and out ratio. However they can be bypassed by executing many independent trades in a single transaction, as it has been done in the latest Balancer attack.

When this price manipulation happens the only trustable sources are the Oracle's price feed, the value function **V** and the token **weights**. **The weighted geometric mean formula uses these parameters to calculate the exact pool token price.**

Gulp attack

Balancer pools contracts have a `gulp` function that absorbs any tokens that have been sent to the contract into the pool. This call modifies the balance of only one token of the pool which also

changes the constant value function **V**.

Under this scenario, **the weighted geometric mean cannot be manipulated and is still an exact solution to calculate the pool token price.**

In other words, calling gulp is like doing a donation to the pool. An example of this attack is described in the section: APPENDIX / Gulp attack example.

Balancer adapter solution

Definitions:

B_i is the current balance of token i
W_i is the weight of token i
SP_i^o is the spot price between any two tokens (in and out)
P_{eth}^i is the price of the pool token i in ether reported by Chainlink
CP_i^o is the spot price between any two tokens using Chainlink prices
T_{pt} is the total supply of the pool token
P_{pt} is the price of the pool token

The solution starts by comparing each spot price that the smart contract reports with the Chainlink spot price to find a deviation.

Balancer Spot Price	Chainlink Price	Price deviation
$SP_i^o = \frac{\frac{B_i}{W_i}}{\frac{B_o}{W_o}}$	$CP_i^o = \frac{P_{eth}^o}{P_{eth}^i}$	$D = \frac{SP_i^o}{CP_i^o}$

If the price deviation is within the accepted limits **(D <= MAX_DEVIATION)**, the arithmetic mean formula will be used:

$$P_{pt} = \frac{\sum_i B_i \cdot P_{eth}^i}{T_{pt}}$$

This formula is very cost effective in terms of gas consumption (see section APPENDIX / Gas profiling). It does not use the constant value function **V** to calculate the price, instead, it just multiplies the tokens' balances by the price in ethers provided by Chainlink in order to get the total amount of ethers and divides it by the pool token supply.

However, if there is a deviation **(D > MAX_DEVIATION)**, the weighted geometric mean formula will be used instead:

$$P^*_{pt} = \frac{V}{T_{pt}} \cdot \left(\frac{1}{\prod_i W_i^{W_i}} \right) \cdot \prod_i {P^i_{eth}}^{W_i}$$

This formula uses all variables that cannot be manipulated by a trade attack: the constant value function **V**, the tokens weights, the Chainlink tokens price feed in ethers and the pool tokens total supply.

There is a proof for a 2-token condition in the APPENDIX/ Weighted Geometric mean math for two tokens. A more generalized proof is described in the attached document Weighted Geometric Mean Math.

Source Code

Smart contract

The smart contract:

- Handles tokens with different decimals.
- Handles tokens that are pegged to ETH.
- Uses Balancer's math functions which have already been audited by Consensys and Trail of bits.
- Uses a matrix approximation to reduce the gas consumption of the fractional exponentiation math function.
- Manages all values in wei format.
- Checks in the constructor if the pool is a shared pool (finalized).
- Supports pools of n tokens but it was tested for 2 and 3 tokens.

Constructor		
Param	**Description**	**Example for MKR/WETH pool**
BPool _pool	Balancer pool address	0x9866772A9BdB4Dc9d2c5a4753e8658B8B0Ca1fC3
bool[] _isPeggedToEth	For each token, true if it is pegged to ether. **	[false, true]
uint8[] _decimals	Number of decimals for each token. **	[18, 18]
ILatestAnswerGetter[] _tokenPriceProviders	Chainlink price aggregators address for each token. **	["0xDa3d675d50fF6C555973C4f0424964e1F6A4e7D3", "0x0000000000000000000000000000000000000000"]
uint256 _priceDeviation	Threshold of spot prices deviation: 10^16 represents a 1% deviation. Must be between 1 and 10^18.	30000000000000000
uint256 _K	K Constant K = 1 / (w1^w1 * .. * wn^wn)	1960130000000000000
uint256 _powerPrecision	Precision for power math function.	100000000
uint256[][] _approximationMatrix	Approximation matrix for gas optimization	[10^1, (10)^0.6, (10)^0.4] [10^2, (10^2)^0.6, (10^2)^0.4] …. [10^20, (10^20)^0.6, (10^20)^0.4] (in wei format)

** Token order determined by Balancer pool method "getFinalTokens".

The most important function is **latestAnswer**:

```
  /**
   * Returns the pool's token price.
   * It calculates the price using Chainlink as an external price source and the pool's tokens balances using the weighted arithmetic mean formula.
   * If there is a price deviation, instead of the balances, it uses a weighted geometric mean with the token's weights and constant value function V.
   */
  function latestAnswer() external view returns (uint256) {
    //Get token balances in ethers
    uint256[] memory ethTotals = new uint256[](tokens.length);
    for (uint256 i = 0; i < tokens.length; i++) {
      ethTotals[i] = getEthBalanceByToken(i);
    }

    if (hasDeviation(ethTotals)) {
      //Calculate the weighted geometric mean
      return getWeightedGeometricMean(ethTotals);
    } else {
      //Calculate the weighted arithmetic mean
      return getArithmeticMean(ethTotals);
    }
  }
```

It checks for a deviation calling `hasDeviation` function. If there is a deviation it calls `getWeightedGeometricMean` function otherwise it calls `getArithmeticMean` function.

Function `hasDeviation`

To compare price deviation D_i^o it uses:

Balancer Spot Price:	Chainlink Price:	Ratio:	Ratio:
$SP_i^o = \frac{\frac{B_i}{W_i}}{\frac{B_o}{W_o}}$	$CP_i^o = \frac{P_{eth}^o}{P_{eth}^i}$	$R = \frac{SP_o^i}{CP_o^i}$	$R = \frac{\frac{B_i \cdot P_{eth}^i}{W_i}}{\frac{B_o \cdot P_{eth}^o}{W_o}}$

There is a price deviation when Ratio > 1 + priceDeviaton or Ratio < 1 - priceDeviaton

Function: `getArithmeticMean`

$$P_{pt} = \frac{\sum_i B_i \cdot P_{eth}^i}{T_{pt}}$$

Function: `getWeightedGeometricMean`

$$P_{pt}^* = \frac{V}{T_{pt}} \cdot \left(\frac{1}{\prod_i W_i^{W_i}} \right) \cdot \prod_i {P_{eth}^i}^{W_i}$$

This formula can be rewritten as

$$P_{pt} = \frac{\prod_i {B_i}^{W_i}}{T_{pt}} \cdot \left(\frac{1}{\prod_i W_i^{W_i}} \right) \cdot \prod_i {P_{eth}^i}^{W_i}$$

Notice that it does not affect the final result to use the balances because they exist in the context of V.

For practical use in the smart contract, we rewrite it in the following format:

$$P_{pt} = \frac{1}{T_{pt}} \cdot \frac{1}{\prod_i W_i^{W_i}} \cdot \prod_i \left(B_i \cdot P_{eth}^i \right)^{Wi}$$

The function approximates the fractional exponentiation using a binomial approximation like Balancer does: https://docs.balancer.finance/protocol/index/approxing

If $B_i \cdot P_{eth}^i$ < 2, it calculates:

$$P_{pt} = \frac{1}{T_{pt}} \cdot \frac{1}{\prod_i W_i^{W_i}} \cdot \prod_i \left(B_i \cdot P_{eth}^i\right)^{Wi}$$

(equation a)

Otherwise, it uses the matrix approximation to calculate:

$$P_{pt} = \frac{1}{T_{pt}} \cdot \frac{1}{\prod_i W_i^{W_i}} \cdot \prod_i \left(c^{W_i} \cdot \left(\frac{B_i \cdot P_{eth}^i}{c} \right)^{Wi} \right)$$

(equation b)

Where **c** is the first bigger constant in the approximation matrix for $B_i \cdot P_{eth}^i$.

(For more information on this matrix approximation, visit the section: APPENDIX / Solidity exponentiation optimization.)

If there is no matrix approximation or the matrix does not contain a constant c >= $B_i \cdot P_{eth}^i$, then to reduce base for the binomial approximation it uses the formula rewritten like this:

$$P_{pt} = \frac{1}{T_{pt}} \cdot \frac{1}{\prod_i W_i^{W_i}} \cdot \prod_i \left(\frac{1}{\left(B_i \cdot P_{eth}^i\right)}^{Wk} \cdot B_i \cdot P_{eth}^i \right)$$

(equation c)

where wk = 1 - wi

It is recommended to add an approximation matrix which has enough constants to use equation b because it uses significantly less gas than equation c.

Deployment

Overview

Each pool to be deployed must be carefully analyzed on a per pool basis to correctly pick the best configuration parameters.
The most important parameters that can optimize a **BalancerSharedPriceProvider** pool to have high precision price calculation and to use the lesser possible gas are:

- Price deviation (*Threshold of spot prices deviation*)
- Power precision (*Precision for power math function*)
- Approximation matrix (*Approximation matrix for gas optimization*)

Price deviation

$$R = \frac{\frac{B_i \cdot P^i_{eth}}{W_i}}{\frac{B_o \cdot P^o_{eth}}{W_o}}$$

There is a price deviation when Ratio > 1 + priceDeviaton or Ratio < 1 - priceDeviaton

A high price deviation could result in using more times the arithmetic mean and saving some gas in certain situations. When picking the 'max price deviation accepted' to use the arithmetic mean, it is easy to think that the % in the price deviation of the spot prices will be similar to the % in price deviation of the pool token price. But in reality the latter is much lower. For more information, please refer to APPENDIX / Precision of arithmetic average formula

Power precision
The less precision used in the fractional exponentiation, the less gas needed. Precision can go from 0 to 18. For a precision X, it must be included in the format of 10^18 / 10^X.
Balancer has it set to a precision of 10:

```
uint public constant BPOW_PRECISION    = BONE / 10**10;
```

Approximation matrix
The more values it has, the less gas the weighted geometric mean may use.
The base constants for the matrix must be bigger than 2, because otherwise they won't be used. A good approximation matrix is one with 20 rows starting from 10^1 to 10^20. This matrix handles balances * price_eth up to 10^20 with no problems.

Example deploy mainnet

This is the Balancer Pool picked:
https://pools.balancer.exchange/#/pool/0x9866772A9BdB4Dc9d2c5a4753e8658B8B0Ca1fC3
It has MKR and WETH with weights of 0.6 and 0.4 respectively.

The deployed BalancerSharedPoolPriceProvider smart contract address is:
https://etherscan.io/address/0xbf809b46e8be63a33b81d9aaa9b252b90f5c7701

It has been deployed with the following params:

Deploy params	
Param	**MKR / WETH pool**
BPool _pool	0x9866772A9BdB4Dc9d2c5a4753e8658B8B0Ca1fC3
bool[] _isPeggedToEth	[false, true]
uint8[] _decimals	[18, 18]
ILatestAnswerGetter[] _tokenPriceProviders	["0xDa3d675d50fF6C555973C4f0424964e1F6A4e7D3", "0x0000000000000000000000000000000000000000"]
uint256 _priceDeviation	30000000000000000
uint256 _K	1960130000000000000
uint256 _powerPrecision	100000000
uint256[][] _approximationMatrix	[10^1, (10)^0.6, (10)^0.4] [10^2, (10^2)^0.6, (10^2)^0.4] …. [10^20, (10^20)^0.6, (10^20)^0.4] (in wei format)

APPENDIX

Precision of arithmetic average formula

$X_i = B_i P_{eth}^i / W_i$ all the X_i approximately equal in normal conditions.

Exact price formula: $\prod_i X_i^{W_i}$ (geometric mean) divided by T_{pt}

Arithmetic mean: $\sum_i W_i X_i = \sum_i B_i P_{eth}^i$

Gas wise cheaper, but what is the error?

Call X_m and X_M the minimum and maximum of (X_i)
If $X_M / X_m < 2$ the error is less than **6.2%,** and always *arithmetic ≥ exact*.

Bound for X_M / X_m (Ratio)	Price deviation	Error
1.5	0.5	2.1%
2	1	6.2%
2.25	1.25	8.5%
2.5	1.5	11%
3	2	16%

Method used: exhaustive test of variables.

Put $X_m = 1$, $X_M =$ *chosen bound*; one (or none) X_i in the middle, with resolution 0.01.
$W_0 + W_1 + W_2 = 1$ all the possibilities with resolution 0.01.

Gas profiling

For 2 Tokens

Price Precision	W1	W2	Arithmetic mean (no manipulation)	Manipulation	Matrix	Geometric mean	Geometric mean without matrix
10 decimals	0.1	0.9	22559 gas	Trade 50%	30 x 3	121415	1476710
8 decimals	0.1	0.9	22559 gas	Trade 50%	30 x 3	116343	1024307
5 decimals	0.1	0.9	22559 gas	Trade 50%	30 x 3	110003	399059
3 decimals	0.1	0.9	22559 gas	Trade 50%	30 x 3	104931	154471
10 decimals	0.1	0.9	22559 gas	Trade 50%	20 x 3	208907	1476710
8 decimals	0.1	0.9	22559 gas	Trade 50%	20 x 3	174671	1024307
5 decimals	0.1	0.9	22559 gas	Trade 50%	20 x 3	130291	399059
3 decimals	0.1	0.9	22559 gas	Trade 50%	20 x 3	110003	127501

For 3 Tokens

Price Precision	W1	W2	W3	Arithmetic mean (no manipulation)	Manipulation	Matrix	Geometric mean	Geometric mean without matrix
10 decimals	0.3	0.3	0.4	32667 gas	Trade 50%	30 x 3	162271	936439
8 decimals	0.3	0.3	0.4	32667 gas	Trade 50%	30 x 3	133107	658642
5 decimals	0.3	0.3	0.4	32667 gas	Trade 50%	30 x 3	91263	287827
3 decimals	0.3	0.3	0.4	32667 gas	Trade 50%	30 x 3	67171	114361
10 decimals	0.3	0.3	0.4	32667 gas	Trade 50%	20 x 3	212207	936439
8 decimals	0.3	0.3	0.4	32667 gas	Trade 50%	20 x 3	169095	658642
5 decimals	0.3	0.3	0.4	32667 gas	Trade 50%	20 x 3	108231	287827
3 decimals	0.3	0.3	0.4	32667 gas	Trade 50%	20 x 3	72727	114361

Solidity exponentiation optimization

For exponentiation, we approximate using a binomial approximation like Balancer does: https://docs.balancer.finance/protocol/index/approxing

$$(1+x)^{\alpha} = 1 + \alpha x + \frac{1}{2}\alpha(\alpha-1)x^2 + \frac{1}{6}\alpha(\alpha-1)(\alpha-2)x^3 + \frac{1}{24}\alpha(\alpha-1)(\alpha-2)(\alpha-3)x^4 + \cdots$$

The closer x gets to 0, the faster it converges to the result and the less gas the smart contract uses.

Approximating x gets to 0 is equal to approximating $(1+x)$ to 1, which means **approximating the base to 1**.

We can define:

$$b^n = c^n \cdot \left(\frac{b}{c}\right)^n$$

where the closer that c is to b, then the closer that $\left(\frac{b}{c}\right)$ is to 1 for $b \geq 1$.

If we apply that to our formula

$$P_{pt} = \frac{1}{T_{pt}} \cdot \frac{1}{\prod_i W_i^{W_i}} \cdot \prod_i \left(B_i \cdot P_{eth}^i\right)^{Wi}$$

$$P_{pt} = \frac{1}{T_{pt}} \cdot \frac{1}{\prod_i W_i^{W_i}} \cdot \prod_i \left(c^{W_i} \cdot \left(\frac{B_i \cdot P_{eth}^i}{c}\right)^{Wi}\right)$$

So we define the constant matrix N x M:

- N is the number of constants
- M number of weights + 1

Example for two tokens

$$\begin{pmatrix} 10 & 10\hat{}w1 & 10\hat{}w2 \\ 100 & 100\hat{}w1 & 100\hat{}w2 \\ 1000 & 1000\hat{}w1 & 1000\hat{}w2 \\ .. & .. & .. \\ 10000000 & 10000000\hat{}w1 & 10000000\hat{}w2 \end{pmatrix}$$

Gulp attack example

Data
Tokens:DAI, MKR
Price DAI: 0.00448268 ETH
Price MKR:1.94656317 ETH
Weights: 0.1 DAI and 0.9 MKR
Pool token supply = 3000 t
Balances for total of 100 ETH

Calculate balances:
DAI: 0.1 * 100 / 0.00448268 = 2230.80
MKR: 0.9 * 100 / 1.94656317 = 46.235

V = 2230.80^0.1 * 46.235^0.9 = 68.1271

Naive approach

$$P_{pt} = \frac{\sum_i B_i \cdot P^i_{eth}}{T_{pt}}$$

Pt = (2230.80 * 0.00448268 + 46.235 * 1.94656317) / 3000 = 100 / 3000 = 0.033333333

Geometric mean approach

$$P^*_{pt} = \frac{V}{T_{pt}} \cdot \left(\frac{1}{\prod_i W_i^{W_i}} \right) \cdot \prod_i {P^i_{eth}}^{W_i}$$

Pt* = (68.1271 / 3000) * (1 /
(0.1^0.1 * 0.9^0.9)) *
0.00448268^0.1 * 1.94656317^0.9 = 0.0333331 (price is the same)

Gulp attack
Add 100 ETH in DAI which is 22308.1 ETH

New Balance for total:
DAI: 2230.80 + 22308.1 = 24538.9
MKR: 46.235

New V = 24538.9^0.1 * 46.235^0.9 = 86.5883

Naive approach

$$P_{pt} = \frac{\sum_i B_i \cdot P_{eth}^i}{T_{pt}}$$

Pt = (24538.9 * 0.00448268 + 46.235 * 1.94656317) / 3000 = 100 / 3000 = 0.0666664 (price manipulated)

Geometric mean approach

$$P_{pt}^* = \frac{V}{T_{pt}} \cdot \left(\frac{1}{\prod_i W_i^{W_i}} \right) \cdot \prod_i P_{eth}^{i\ W_i}$$

Pt* = (86.5883 / 3000) * (1 /
(0.1^0.1 * 0.9^0.9)) *
0.00448268^0.1 * 1.94656317^0.9 = 0.0423658 (real new price)

Cost to buy back DAI and decrease price

$$A_i = B_i \cdot \left(\left(\frac{B_o}{B_o - A_o} \right)^{\frac{W_o}{W_i}} - 1 \right)$$

Mkr to spend = 46.235 * (((24538.9 / (24538.9 - 22308.1)) ^ (0.1/0.9) - 1)
Mkr to spend = 14.1156 mkr
ETH to spend = 14.1156 * 1.94656317 = 27.47 ETH

Total ETH simulated using weighted geometric mean
3000 * 0.0423658 = 127.0974 ETH

To simulate he has 127.0974 he needs to spend 27.47 ETH

Weighted Geometric Mean math for two tokens

Know equations:

$$(B_1)^{W_1} . (B_2)^{W_2} = V$$ (eq 1)

$$SP_1^2 = \frac{\frac{B_1}{W_1}}{\frac{B_2}{W_2}}$$ (eq 2)

$$SP_1^2 = \frac{P_{eth}^2}{P_{eth}^1}$$ (eq 3)

$$T_{eth} = B_1 \cdot P_{eth}^1 + B_2 \cdot P_{eth}^2$$ (eq 4)

$$P_{pt} = \frac{T_{eth}}{T_{pt}}$$ (eq 5)

W1 + W2 = 1 (eq 6)

Using eq 2 :

$$SP_1^2 = \frac{\frac{B_1}{W_1}}{\frac{B_2}{W_2}} \qquad SP_1^2 = \frac{B_1 \cdot W_2}{B_2 \cdot W_1} \qquad B_1 = \frac{B_2 \cdot W_1 \cdot SP_1^2}{W_2}$$ (eq 7)

Using eq 3 and eq 7:

$$B_1 = \frac{B_2 \cdot W_1 \cdot \frac{P^2_{eth}}{P^1_{eth}}}{W_2} \qquad \boxed{B_1 = \frac{B_2 \cdot W_1 \cdot P^2_{eth}}{W_2 \cdot P^1_{eth}}} \text{ (eq 8)}$$

Using eq 1, eq 6 and eq 8

$$(B_1)^{W_1} . (B_2)^{W_2} = V \qquad \left(\frac{B_2 \cdot W_1 \cdot P^2_{eth}}{W_2 \cdot P^1_{eth}} \right)^{W_1} . (B_2)^{W_2} = V$$

$$B_2^{W_1+W_2} \cdot \left(\frac{W_1 \cdot P^2_{eth}}{W_2 \cdot P^1_{eth}} \right)^{W_1} = V \qquad \boxed{B_2 = \frac{V}{\left(\frac{W_1 \cdot P^2_{eth}}{W_2 \cdot P^1_{eth}} \right)^{W_1}}} \text{ (eq 9)}$$

Same steps for B1 you get

$$\boxed{B_1 = \frac{V}{\left(\frac{W_2 \cdot P^1_{eth}}{W_1 \cdot P^2_{eth}} \right)^{W_2}}} \text{eq 10}$$

Using eq 4, 6, 9, 10

$$T_{eth} = B_1 \cdot P^1_{eth} + B_2 \cdot P^2_{eth}$$

$$T_{eth} = \frac{V}{\left(\frac{W_2 \cdot P^1_{eth}}{W_1 \cdot P^2_{eth}}\right)^{W_2}} \cdot P^1_{eth} + \frac{V}{\left(\frac{W_1 \cdot P^2_{eth}}{W_2 \cdot P^1_{eth}}\right)^{W_1}} \cdot P^2_{eth}$$

$$T_{eth} = V \cdot \left({W_2}^{-W_2} \cdot {P^1_{eth}}^{-W_2} \cdot {W_1}^{W_2} \cdot {P^2_{eth}}^{W_2} \right) \cdot P^1_{eth} + \left({W_1}^{-W_1} \cdot {P^2_{eth}}^{-W_1} \cdot {W_2}^{W_1} \cdot {P^1_{eth}}^{W_1} \right) \cdot P^2_{eth}$$

$$T_{eth} = V \cdot \left({W_2}^{-W_2} \cdot {P^1_{eth}}^{1-W_2} \cdot {W_1}^{1-W_1} \cdot {P^2_{eth}}^{W_2} \right) + \left({W_1}^{-W_1} \cdot {P^2_{eth}}^{1-W_1} \cdot {W_2}^{1-W_2} \cdot {P^1_{eth}}^{W_1} \right)$$

$$T_{eth} = V \cdot \left({W_2}^{-W_2} \cdot {P^1_{eth}}^{W_1} \cdot {W_1}^{1-W_1} \cdot {P^2_{eth}}^{W_2} \right) + \left({W_1}^{-W_1} \cdot {P^2_{eth}}^{W_2} \cdot {W_2}^{1-W_2} \cdot {P^1_{eth}}^{W_1} \right)$$

$$T_{eth} = V \cdot {W_2}^{-W_2} \cdot {W_1}^{-W_1} \cdot {P^2_{eth}}^{W_2} \cdot {P^1_{eth}}^{W_1} (\cdot W_1 + \cdot W_2)$$

$$T_{eth} = V \cdot {W_2}^{-W_2} \cdot {W_1}^{-W_1} \cdot {P^2_{eth}}^{W_2} \cdot {P^1_{eth}}^{W_1}$$

$$T_{eth} = V \cdot \frac{1}{{W_2}^{W_2} \cdot {W_1}^{W_1}} \cdot {P^2_{eth}}^{W_2} \cdot {P^1_{eth}}^{W_1}$$ eq 11

Using eq 5 and eq 11

$$P_{pt} = \frac{T_{eth}}{T_{pt}}$$

$$P_{pt} = \frac{V}{T_{pt}} \cdot \frac{1}{{W_2}^{W_2} \cdot {W_1}^{W_1}} \cdot {P^2_{eth}}^{W_2} \cdot {P^1_{eth}}^{W_1}$$

eq 12

Eq12 is the price of the pool token for a pool with 2 tokens using the trustable datasources: V, weights and price of each token in ethers from Chainlink Oracle.