Uniswap V2 Price Provider

Specification 1.0 / 05-08-2020

Summary **2**

Requirements **2**

Price manipulation **2**

Trade attack 2

Sync attack 2

Uniswap V2 adapter solution **3**

Source Code **4**

Overview 4

Smart contract 4

LatestAnswer function 5

GetTotalSupplyAtWithdraw function 7

Deployment **8**

Example deploy mainnet 8

APPENDIX **9**

Weighted Geometric Mean Proof 9

Summary

This document describes the implementation of a Uniswap V2 adapter (Price Provider) to get the price in ether for a pair token also known as UNI-V2 Token.

Requirements

The implementation of the Uniswap V2 adapter must fulfill this requirements:

- Pair token price cannot be manipulated
- Chainlink will be used as the main oracle
- It should use as less gas as possible

Price manipulation

After carefully reading the Uniswap V2 Whitepaper, audit and smart contracts, we have identified two potential attacks that could manipulate the pair token price: trade attack and sync attack.

Trade attack

Uniswap V2 pairs 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 pair's total value in ethers. This might result in an inflated price of the UNI-V2 token.

When this price manipulation happens the only trustable sources are the Oracle's price feed and the invariant **K**. **The weighted geometric mean formula uses these parameters to calculate the exact pair token price.**

Sync attack

Uniswap V2 pairs contracts have a `sync` function that absorbs any tokens that have been sent to the contract into the pair. This call modifies the balance of only one token of the pair which also changes invariant **K**.

Under this scenario, **the weighted geometric mean <u>cannot be manipulated</u> and is still an exact solution to calculate the pair token price.**

In other words, calling sync is like doing a donation to the pair.

Uniswap V2 adapter solution

<u>Definitions</u>:

$B_i^{\square}$ is the current balance 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 pair token i in ether reported by Chainlink

CP_i^o is the spot price between any two tokens using Chainlink prices

$T_{pt}^{\square}$ is the total supply of the pair token

$P_{pt}^{\square}$ is the price of the pair token

Solution

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

Uniswap V2 Spot Price	Chainlink Price	Ratio
$SP_i^o = \frac{B_i}{B_o}$	$CP_i^o = \frac{P_{eth}^o}{P_{eth}^i}$	$R = \frac{SP_o^i}{CP_o^i} = \frac{B_i \cdot P_{eth}^i}{B_o \cdot P_{eth}^o}$

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

$$P_{pt} = \frac{B_1 \cdot P_{eth}^1 + B_2 \cdot P_{eth}^2}{T_{pt}}$$

This formula does not use the invariant **K** 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 pair token supply.

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

$$P_{pt} = \frac{2 \cdot \sqrt{K \cdot P_{eth}^1 \cdot P_{eth}^2}}{T_{pt}}$$

This formula uses all variables that cannot be manipulated by a trade attack: the invariant **K** and the Chainlink tokens price feed in ethers and the pair tokens total supply

Source Code

The smart contract:

- Handles tokens with different decimals.
- Handles tokens that are pegged to ETH.
- Manages all values in wei format.
- Handles protocol fee if activated.

Constructor		
Param	**Description**	**Example for MKR/WETH pair**
IUniswapV2Pair _pair	Uniswap V2 pair address	0xC2aDdA861F89bBB333c90c492cB837741916A225
bool[] _isPeggedToEth	For each token, true if it is pegged to ether. **	[false, true]
uint8[] _decimals	Number of decimals for each token. **	[18, 18]
IPriceOracle _priceOracle	Aave price oracle.	0x76B47460d7F7c5222cFb6b6A75615ab10895DDe4
uint256 _priceDeviation	Threshold of spot prices deviation: 10ˆ16 represents a 1% deviation. Must be between 1 and 10ˆ18.	30000000000000000

** Token order is sorted by its address.

LatestAnswer function

The most important function is **lastestAnswer**:

```
 /**
* Returns the pair's token price.
* It calculates the price using Chainlink as an external price source and the pair's tokens reserves using the arithmetic mean formula.
* If there is a price deviation, instead of the reserves, it uses a weighted geometric mean with constant invariant K.
*/
function latestAnswer() external view returns (uint256) {
   //Get token reserves in ethers
   (uint112 reserve_0, uint112 reserve_1, ) = pair.getReserves();
   uint256 ethTotal_0 = getEthBalanceByToken(0, reserve_0);
   uint256 ethTotal_1 = getEthBalanceByToken(1, reserve_1);

   if (hasDeviation(ethTotal_0, ethTotal_1)) {
      //Calculate the weighted geometric mean
      return getWeightedGeometricMean(ethTotal_0, ethTotal_1);
   } else {
      //Calculate the arithmetic mean
      return getArithmeticMean(ethTotal_0, ethTotal_1);
   }
}
```

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 ratio R it uses:

Uniswap V2 Spot Price:	Chainlink Price:	Ratio:	Ratio:
$SP_i^o = \frac{B_i}{B_o}$	$CP_i^o = \frac{P_{eth}^o}{P_{eth}^i}$	$R = \frac{SP_o^i}{CP_o^i}$	$R = \frac{B_i \cdot P_{eth}^i}{B_o \cdot P_{eth}^o}$

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

Function: `getArithmeticMean`

$$P_{pt} = \frac{B_1 \cdot P_{eth}^1 + B_2 \cdot P_{eth}^2}{T_{pt}}$$

Function: `getWeightedGeometricMean`

$$P_{pt} = \frac{2 \cdot \sqrt{K \cdot P_{eth}^1 \cdot P_{eth}^2}}{T_{pt}}$$

This formula can be rewritten as

$$P_{pt} = \frac{2 \cdot \sqrt{B_1 \cdot P_{eth}^1 \cdot B_2 \cdot P_{eth}^2}}{T_{pt}}$$

GetTotalSupplyAtWithdraw function

Uniswap V2 collects 0.05% fee when liquidity is deposited or withdrawn. The contract computes the accumulated fees, and mints new liquidity tokens to the fee beneficiary, immediately before any tokens are minted or burned. The total collected fees are computed using this formula:

$$s_m = \frac{\sqrt{k_2} - \sqrt{k_1}}{5 \cdot \sqrt{k_2} + \sqrt{k_1}} \cdot s_1$$

For more information about the formula, please refer to Uniswap V2 whitepaper.

Currently this fee is not activated, but this can change at any moment and will directly affect the total supply of the pair token and thus the arithmetic and weighted geometric mean formulas.

To deal with this change in total supply, the price provider contract not only checks the total supply of the pair but it also computes how much will be minted at the time of withdrawal if the fee is activated. This is done in the **getTotalSupplyAtWithdrawal** function:

```
/**
   * Returns Uniswap V2 pair total supply at the time of withdrawal.
   */
  function getTotalSupplyAtWithdrawal() private view returns (uint256 totalSupply) {
     totalSupply = pair.totalSupply();
     address feeTo = IUniswapV2Factory(IUniswapV2Pair(pair).factory()).feeTo();
     bool feeOn = feeTo != address(0);
     if (feeOn) {
        uint kLast = IUniswapV2Pair(pair).kLast();
        if (kLast != 0) {
           (uint112 reserve_0, uint112 reserve_1, ) = pair.getReserves();
           uint256 rootK = Math.bsqrt(uint256(reserve_0).mul(reserve_1), false);
           uint256 rootKLast = Math.bsqrt(kLast, false);
           if (rootK > rootKLast) {
              uint256 numerator = totalSupply.mul(rootK.sub(rootKLast));
              uint256 denominator = rootK.mul(5).add(rootKLast);
              uint256 liquidity = numerator / denominator;
              totalSupply += liquidity;
           }
        }
     }
  }
```

APPENDIX

Weighted Geometric Mean Proof

Know equations:

$$\boxed{x \cdot y = K}$$ (eq 0)

$$\boxed{P_{pt} = \frac{x \cdot P^x_{eth} + y \cdot P^y_{eth}}{T_{pt}}}$$ (eq 1)

$$\boxed{SP^y_x = \frac{y}{x}}$$ (eq 2)

$$\boxed{CP^y_x = \frac{P^x_{eth}}{P^y_{eth}}}$$ (eq 3)

From eq 3 and 2:

$$\frac{y}{x} = \frac{P^x_{eth}}{P^y_{eth}}$$ (eq 4)

From eq 4:

$$x = \frac{P^{y}_{eth} \cdot y}{P^{x}_{eth}}$$ (eq 5)

From y is the same:

$$y = \frac{P^{x}_{eth} \cdot x}{P^{y}_{eth}}$$ (eq 6)

From eq 0 and 6:

$$K = x \cdot \frac{P^{x}_{eth} \cdot x}{P^{y}_{eth}}$$ (eq 7)

From eq 7:

$$\boxed{x = \sqrt{K \cdot \frac{P^{y}_{eth}}{P^{x}_{eth}}}}$$ (eq 8)

Same for y:

$$\boxed{y = \sqrt{K \cdot \frac{P^{x}_{eth}}{P^{y}_{eth}}}}$$ (eq 9)

From eq 1, 8 and 9:

$$P_{pt} = \frac{\sqrt{K \cdot \frac{P^y_{eth}}{P^x_{eth}}} \cdot P^x_{eth} + \sqrt{K \cdot \frac{P^x_{eth}}{P^y_{eth}}} \cdot P^y_{eth}}{T_{pt}}$$

(eq 10)

From eq 10:

$$P_{pt} = \frac{K^{0.5} \cdot {P^y_{eth}}^{0.5} \cdot {P^x_{eth}}^{-0.5} \cdot P^x_{eth} + K^{0.5} \cdot {P^x_{eth}}^{0.5} \cdot {P^y_{eth}}^{-0.5} \cdot P^y_{eth}}{T_{pt}}$$

(eq 11)

From eq 11:

$$P_{pt} = \frac{K^{0.5} \cdot {P^y_{eth}}^{0.5} \cdot {P^x_{eth}}^{0.5} + K^{0.5} \cdot {P^x_{eth}}^{0.5} \cdot {P^y_{eth}}^{0.5}}{T_{pt}}$$

(eq 12)

From eq 12:

$$\boxed{P_{pt} = \frac{2 \cdot \sqrt{K \cdot P^x_{eth} \cdot P^y_{eth}}}{T_{pt}}}$$

(eq 13)

Eq13 is the price of the UNI-V2 token for a pair using the trustable datasources: invariant K and price of each token in ethers from Chainlink Oracle.