Price Oracle

**Introduction**

Oracles are central points of failure in any smart contract system. Any major deviation in prices can severely impact trust in Opyn.

The specification is an attempt to expand Opyn’s existing oracle system in addition to Chainlink to have a more robust oracle system.

The document’s intended audience is Opyn’s engineering team.

**Task Overview**

Opyn v2 uses cash settled options. It means that after an option expires, Opyn needs to be able to access exact price of the underlying asset at the expiry timestamp. Currently Chainlink is used as the oracle to give us the historical price at expiry for an asset. However, relying on Chainlink can lead to a central point of failure.

This proposed solution to the above problem is to utilize Uniswap’s TWAP based Oracle to provide a complement alterative to the existing Chainlink Oracle.

**Design Considerations**

In selecting the price Oracle solution to use, two options are looked at:

* UMA’s DVM (Data Verification Mechanism) Oracle
* TWAP on Uniswap V2

[UMA’s DVM Oracle](https://docs.umaproject.org/oracle/dvm-interface)

* The DVM is powered by voters. Each price request from a financial contract must be voted on. The relative weighting of votes is determined by how many VoteTokens each participant holds. – Pro: decentralized oracle.
* “A financial contract should use requestPrice (function) whenever it needs a price from the DVM. Generally, it should only be used as an arbitration mechanism to resolve disputes and to settle risk. If a financial contract template overuses this function, it's unlikely to be approved for use with the DVM.” Con: not fitted for frequent price lookup.
* UMA have voters (commit and reveal) provide price. Price votes can be delayed coming in for asset price lookup request in its queue due to network congestion. Con: performance affected by network congestion.
* It is possible that a request will have to be delayed to a future voting round when less than minimal required number of price votes are submitted due to network delay or lack of participation from the voters. Con: stale price info due to insufficient votes.

[Uniswap TWAP (Time Weighted Average Price) Oracle](https://uniswap.org/docs/v2/core-concepts/oracles/)

* A time-weighted oracle service (time-weighting makes shenanigans expensive) that keeps price-time data recorded on-chain in a way that is expensive to manipulate over small periods of time, and impossible to manipulate within a single transaction.
* Using “cumulative” price-time values, how long a price is available is weighted into a special value, with every token swap spending a small amount of gas to keep these values in sync.
* The average price depends on prices that appeared in the past, proportionally to how long they appeared for, and the oracle consumer can choose the length of the period for averaging.
* The average price is not influenced by prices the appeared within a block, but only by the final price at the end of a block.
* For a simple TWAP, the cost of manipulation increases (approx. linear) with liquidity on Uniswap, as well as (approx. linear) with the length of time over which you average. -potential con – cost variation
* There is a separate exchange contract for every ERC20 token. The getExchange method in the factory contract can be used to find the Ethereum address associated with an ERC20 token address.

**Assumptions**

Assuming frequent Oracle lookup is necessary for option cash payout and collateral assessment by Opyn, accurate/fresh price info is key for Opyn’s Oracle system.

**Constraints and Restrictions**

[Per Uniswap documentation](https://uniswap.org/docs/v2/smart-contract-integration/building-an-oracle/),

* for Uniswap oracle to measure average prices over a period, the oracle must have a way of referencing the cumulative price at the start and end of a period. The recommended way of doing this is by storing these prices in the oracle contract and calling the oracle frequently enough to store the latest cumulative price.
* Reliable oracle maintenance is a difficult task and can become a point of failure in times of congestion.

**Dependencies**

Install Keydonix packages –

npm install @keydonix/uniswap-oracle-contracts @keydonix/uniswap-oracle-sdk

**Design**

* UMA’s DVM Oracle is not selected after evaluation as it is recommended as arbitration mechanism to resolve disputes and to settle risk, and its accuracy may get affected by voting delays due to network congestion or lack of voter participation.
* Uniswap’s TWAP Oracle is the selected Oracle option for this task.
* To avoid regularly storing of available cumulative prices, [Keydonix](•%09https:/github.com/Keydonix/uniswap-oracle) is used for its general purpose price feed oracle built on Uniswap v2 that supports arbitrary time windows (up to 256 blocks) and doesn’t require any active maintenance. (Note that the greatest number of past blocks being queried for past cumulative price is 256.)
* Introduced 2 smart contracts and 1 interface –
  + **UniswapPricer.sol**

A pricer contract that implements OpynPricerInterface, a pricer counterpart of ChainlinkPricer.

Key functions –

getPrice() external view returns (uint256);

1. gets the live price for the asset.
2. overides the getPrice function in OpynPricerInterface
3. returns price of the asset in USD, scaled by 1e8

setExpiryPriceInOracle (uint256 \_expiryTimestamp, uint256 \_blockNumber) external onlyBot

1. set the expiry price in the oracle, can only be called by Bot address

Variables –

/// the opyn oracle address

OracleInterface public oracle;

/// the uniswap adapter for an asset

UniswapAdapterInterface public adapter;

/// asset that this pricer will a get price for

address public asset;

/// bot address that is allowed to call setExpiryPriceInOracle

address public bot;

///Uniswap v2 pair involving the asset

IUniswapV2Pair public uniswapV2Pair;

///denomination token for the asset above in V2 pair

address public denominationToken;

///minimal number of blocks before the current block

uint8 public minBlocksBack;

///maximal number of blocks before the current block

uint8 public maxBlocksBack;

///storage proof data

UniswapOracle.ProofData public proofData;

* + **UniswapAdapter**

Inherited external contract UniswapOracle from Keydonix implementation to provide non-maintenance oracle.

Key functions –

getPrice(IUniswapV2Pair \_exchange, address \_denominationToken, uint8 \_minBlocksBack, uint8 \_maxBlocksBack, UniswapOracle.ProofData memory \_proofData) public returns (uint256 price) ;

1. get the live price for the asset via UniswapOracle.getPrice()
2. \_uniswapV2Pair Uniswap v2 pair involving the asset corresponding to \_expiryTimestamp
3. \_denominationToken denomination token for the asset in the pair
4. \_minBlocksBack minimal number of blocks before the current block
5. \_maxBlocksBack maximal number of blocks before the current block
6. \_proofData storage proof data
7. return price of the asset in USD, scaled by 1e8

getPriceForSpecificTimePoint(IUniswapV2Pair \_uniswapV2Pair, address \_denominationToken, uint8 \_minBlocksBack, uint8 \_maxBlocksBack, ProofData memory \_proofData, uint256 \_expiryTimestamp) public view returns (uint256 price) ;

1. get price for the asset at for a specific \_expiryTimestamp
   * **interface UniswapAdapterInterface**

Key functions –

getPrice(IUniswapV2Pair \_exchange, address \_denominationToken, uint8 \_minBlocksBack, uint8 \_maxBlocksBack, UniswapOracle.ProofData memory \_proofData) public returns (uint256 price) ;

getPriceForSpecificTimePoint(IUniswapV2Pair \_uniswapV2Pair, address \_denominationToken, uint8 \_minBlocksBack, uint8 \_maxBlocksBack, ProofData memory \_proofData, uint256 \_expiryTimestamp) public view returns (uint256 price) ;

getPriceRaw(IUniswapV2Pair \_uniswapV2Pair, bool \_denominationTokenIs0, uint8 \_minBlocksBack, uint8 \_maxBlocksBack, ProofData memory \_proofData, uint256 \_expiryTimestamp) public view returns (uint256 price) ;

getSpecificTimePriceCumulativeLast(IUniswapV2Pair \_uniswapV2Pair, bool \_denominationTokenIs0, uint256 \_expiryTimestamp) public view returns (uint256 priceCumulativeLast) ;

**Bibliography**

* <https://docs.google.com/document/d/13xDJoPqfR6H3N_AMdBuSV8t20pEGiUqo4kCOJkBEV3o/edit#heading=h.9jlxs4h1ulox>
* <https://github.com/opynfinance/GammaProtocol/blob/master/contracts/Oracle.sol>
* <https://uniswap.org/docs/v2/core-concepts/oracles/>
* <https://docs.umaproject.org/getting-started/oracle>
* <https://github.com/Keydonix/uniswap-oracle>
* <https://docs.umaproject.org/oracle/known-issues>