Delv Hyperdrive Security Analysis Report and Formal Verification Properties

x

08.08.2023

Table of Contents

Summary

This document describes the specification and verification of the new **Delv Hyperdrive protocol** using the Certora Prover and manual code review findings. The work was undertaken from **03rd May 2023** to **01st July 2023**. The commits reviewed and run through the Certora Prover were from 9e960c55 to 4827596c. The latest commit that was reviewed manually is 7233c4d8.

The following contracts list is included in the **scope**:

```
contracts/src/factory/AaveHyperdriveDeployer.sol
contracts/src/factory/DsrHyperdriveDeployer.sol
contracts/src/factory/ERC4626HyperdriveDeployer.sol
contracts/src/factory/HyperdriveFactory.sol
contracts/src/factory/StethHyperdriveFactory.sol
contracts/src/factory/AaveHyperdriveFactory.sol
contracts/src/factory/DsrHyperdriveFactory.sol
contracts/src/factory/ERC4626HyperdriveFactory.sol
contracts/src/factory/StethHyperdriveDeployer.sol
contracts/src/instances/AaveHyperdrive.sol
contracts/src/instances/DsrHyperdrive.sol
contracts/src/instances/ERC4626DataProvider.sol
contracts/src/instances/StethHyperdrive.sol
contracts/src/instances/AaveHyperdriveDataProvider.sol
contracts/src/instances/DsrHyperdriveDataProvider.sol
contracts/src/instances/ERC4626Hyperdrive.sol
contracts/src/instances/StethHyperdriveDataProvider.sol
contracts/src/libraries/AssetId.sol
contracts/src/libraries/Errors.sol
contracts/src/libraries/FixedPointMath.sol
contracts/src/libraries/HyperdriveMath.sol
contracts/src/libraries/SafeCast.sol
contracts/src/libraries/YieldSpaceMath.sol
contracts/src/token/BondWrapper.sol
contracts/src/token/ERC20Forwarder.sol
contracts/src/token/ForwarderFactory.sol
contracts/src/token/MultiToken.sol
contracts/src/token/MultiTokenDataProvider.sol
contracts/src/token/MultiTokenStorage.sol
contracts/src/DataProvider.sol
contracts/src/HyperdriveDataProvider.sol
contracts/src/HyperdriveShort.sol
contracts/src/Hyperdrive.sol
contracts/src/HyperdriveLP.sol
```

`contracts/src/HyperdriveStorage.sol`
`contracts/src/HyperdriveBase.sol`
`contracts/src/HyperdriveLong.sol`
`contracts/src/HyperdriveTWAP.sol`

The contracts are written in Solidity 0.8.19.

The Certora Prover demonstrated that the implementation of the Solidity contracts above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of all Solidity contracts. During the verification process and the manual audit, the Certora Prover discovered bugs in the Solidity contracts code, as listed below.

Summary of findings

The table below summarizes the issues discovered during the audit, categorized by severity.

Severity	**Total discovered**	**Total fixed**	**Total acknowledged**
High	4	4	4
Medium	9	8	9
Low	6	6	6
Informational	24 + gas optimizations	Everything with respect to the new instances implementation	24 + gas optimizations
Total (High, Medium, Low)	**19**	**19**	**19**

Disclaimer

The Certora Prover takes a contract and a specification as input and formally proves that the contract satisfies the specification in all scenarios. Notably, the guarantees of the Certora Prover are scoped to the provided specification and the Certora Prover does not check any cases not covered by the specification.

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

Main Issues Discovered

High-01: Unsafe use of `transfer()`/`transferFrom()` with `IERC20`

Severity: High
Probability: High
Category: DOS / Funds are locked
File(s): BondWrapper.sol
Bug description: Some tokens (like BNB or USDT) do not implement the ERC20 standard properly but are still accepted by most code that accepts ERC20 tokens.
Example:
Tether (USDT)'s `transfer()` and `transferFrom()` functions on L1 do not return booleans as the specification requires, and instead have no return value: live example. When these types of tokens are cast to IERC20, their function signatures do not match and therefore, calls made will revert. Consider using OpenZeppelin's SafeERC20's `safeTransfer()`/`safeTransferFrom()` instead as SafeERC20 ensures consistent handling of ERC20 return values and abstract over inconsistent ERC20 implementations.
For more information, including POC, see here.
Exploit scenario:
If USDT is chosen as the underlying token in BondWrapper, traders successfully use the `mint()` function to wrap their long positions. At maturity, traders try to `close()` their positions or `redeem()` their tokens, but this is impossible, as calls to `token.transfer()` always revert.
Implications: Funds are stuck and impossible to recover.
Delv's response: acknowledged and **fixed**:
https://github.com/Delv/hyperdrive/pull/393.

High-02: First depositor can break minting of shares

Severity: High
Probability: Medium
Category: Donation attack
File(s): DSRHyperdrive.sol
Bug description: Users may not receive shares in exchange for their deposits if the total asset amount has been manipulated through a large "donation". Here, the attempt to supply liquidity can either revert (like in the POC below) or introduce a significant rounding-down impact (at a loss for the liquidity provider).
Exploit scenario / Numerical Example:
Pre-condition:
Considering the DSRHyperdrive Instance, the pool gets initialized with a minimal contribution (2 units of DAI). As deploying official instances through the factory isn't access-controlled, the scenario could be the following. If the deployer doesn't want to think about adding liquidity at first and just sends enough to avoid the zero-amount check.
Happy Scenario:

- The pool gets initialized with a minimal contribution (2 units of DAI). Now `totalBase = 1` and `totalShares = 2`
- Alice (a protocol user) calls `addLiquidity()` with 1000 DAI
  - `newShares = totalShares.mulDivDown(amount, totalBase)`
  - `newShares = 2.mulDivDown(10000000000000000000, 1)`
  - `newShares = 20000000000000000000`
  - Final `totalShares = 20000000000000000002`
- Alice receives her share.

Frontrunning Scenario:

- The pool gets initialized with a minimal contribution (2 units of DAI). Now `totalBase = 1` and `totalShares = 2`.
- Bob monitors the mempool and sees that Alice wants to provide 1000 DAI worth of liquidity to the instance.
- Bob frontruns Alice with a call to `dsrManager.join()` with 2000.01 DAI.
- Now `totalBase = 2000010000000000000000` and `totalShares = 2`.
- Alice calls `addLiquidity()` with 1000 DAI
  - `newShares = totalShares.mulDivDown(amount, totalBase)`
  - `newShares = 2.mulDivDown(10000000000000000000, 2000010000000000000000) = 0.`
  - The `_deposit()` function tries to `return (newShares, amount.divDown(newShares));`, which contains a division by 0 error as `newShares == 0`.
- The transaction reverts.

Implications: An attacker can DOS or introduce losses for liquidity providers under certain deployment conditions.
Delv's response: acknowledged and **fixed**:
https://github.com/Delv/hyperdrive/pull/380.

High-03: Unsafe reentrancy in `StethHyperdrive._deposit()`

Severity: Medium
Probability: High
Category: Reentrancy / losing control of the flow
File(s): StethHyperdrive.sol
Bug description: `StethHyperdrive._deposit()` is vulnerable to reentrancy. Currently, the impact seems equivalent to consecutive calls. However, given the complexity of the code, an attacker with more time could quite possibly find a vector to influence the different state variables that are getting updated after the line where the control flow is given away. This would raise the exploit's impact. Consider adding a `nonReentrant` modifier on all of the contract's entry points. Further information, including POC, can be found here.
Implications: Open attack vector for future exploits if a way is found to manipulate the state variables used after the re-entrance line.
Delv's response: acknowledged and **fixed**: https://github.com/Delv/hyperdrive/pull/455.

High-04: Units mismatch

Severity: Medium
Probability: High
Category: Logic error
File(s): HyperdriveMath.sol
Bug description: The netCurveTrade has units of [bonds], but `maxCurveTrade` has units of [shares] or [bonds/price]. So in the 'else' branch where the net curve trade is negative, there seems to be a mixture of units, and the value that is passed to the `calculateSharesInGivenBondsOut()` should be in bonds, not shares.
Delv's response: acknowledged and **fixed**: https://github.com/Delv/hyperdrive/pull/304.

Medium-01: `close()` in BondWrapper can be sandwiched

Severity: Medium
Probability: Low
Category: Slippage Attack
File(s): BondWrapper.sol
Bug description: When closing a wrapped position, BondWrapper calls `closeLong()` without slippage control (`_minAmount = 0`). This is a common vulnerability that opens the potential for a sandwich attack by MEV bots.
Exploit scenario:

1. Alice closes her wrapped position.
2. MEV bot sandwiches her transaction.

Implications: Loss of funds for the trader.
Delv's response: acknowledged and **fixed**: https://github.com/Delv/hyperdrive/pull/298.

Medium-02: Checkpoints boundary trading (+ systemic bias in accounting)

Severity: High
Probability: Low
Category: Calculation Flaw
File(s): HyperdriveLong.sol
Bug description: When opening a long position, the system doesn't properly handle the time on the bond. It assumes the time remaining is 1.0 at all times. While this is theoretically correct (new bonds should be 0% matured), it doesn't account for checkpoint duration.
Numerical Example / Exploit scenario:

1. Deployer deploys Hyperdrive with `checkpointDuration = 1 day` and `positionDuration = 7` days and very low fees.
2. A trader waits till the end of the current checkpoint and opens a large long.
3. The trader waits for the transaction to be minted.
4. In the next checkpoint, their position will be 14.3% matured (1/7).
5. The trader immediately closes his large position with profit.

Implications: Malicious trading activity can occur if deployment parameters aren't chosen carefully.
Delv's response: *The bug is exploitable only with certain deployment parameters. With reasonable parameters it's not significant (implementing a 100% fair fix makes code more complicated). During deployment there should be sanity checks for system parameters.*
Specifically:

- *fees should be reasonably high.*
- *the ratio between position_duration and checkpoint duration should be high".*

Medium-03: A minimally deployed DSRHyperdrive instance cannot receive liquidity

Severity: Medium
Probability: Low
Category: Unexpected Self-DOS
File(s): DSRHyperdrive.sol
Bug description: When initializing a pool with a minimal contribution (1 unit of DAI), it will be impossible to add liquidity to the system by calling `addLiquidity()` or `openLong()` due to the `_deposit()` function reverting with a divide-by-zero error.
Exploit scenario / Numerical Example:
Pre-condition:
We are on the `DSRHyperdrive.sol` instance. The pool gets initialized with a minimal contribution (1 unit of DAI). Deploying official instances through the factory is not access-controlled. Therefore, this issue could happen, for example, when the deployer doesn't want to think about adding liquidity at first and just sends enough to avoid the zero-amount check.
Scenario:
The pool gets initialized with a minimal contribution (1 unit of DAI). Now `totalBase = 0` and `totalShares = 1`
Alice decides to supply liquidity to the instance and calls `addLiquidity()` with 1000 DAI.
Given that `totalShares != 0`, the following expression will be evaluated:
`uint256 newShares = totalShares.mulDivDown(amount, totalBase);`
however, `totalBase == 0`.
The transaction reverts due to a divide-by-zero error.
The same happens by calling `openLong()`.
Implications: The instance cannot be used/supplied with liquidity.
Delv's response: acknowledged and **fixed**:
https://github.com/Delv/hyperdrive/pull/380.

Medium-04: `ERC4626Hyperdrive.sweep()` can sweep the underlying token if there are multiple entry points

Severity: High
Probability: Low
Category: Rug Scenario
File(s): ERC4626Hyperdrive.sol
Bug description: A new `sweep()` function has been added in the PR.
While it does contain a check against the pool token or the `_baseToken`, there can still be an issue/rug scenario when the token has multiple entry points, meaning there are two different contracts that are both interacting with the same balances.
This type of vulnerability can occur, as an example, when a contract enables a secondary entry point when upgrading its contract (like with TrueUSD or sUSD).
This can also happen in the future with upgradeable contracts like Tether USD (USDT).
Implications: Rug vector on the underlying token.
Delv's response: acknowledged and **fixed**:
https://github.com/Delv/hyperdrive/pull/473.

Medium-05: `StethHyperdrive.sol`: Denial of Service when LIDO's `TotalPooledEther` decreases.

Severity: High
Probability: Low
Category: DOS
File(s): StethHyperdrive.sol
Bug description: When LIDO decreases its `TotalPooledEther` balance, such as when by updating the Consensus Layer's state snapshot (`CL_BALANCE_POSITION`), this will have an effect on prices and shares.
Such an update (as small as 1 Ether, meaning 1e18), will decrease `_shareProceeds` and increase `_shareReservesDelta`, in such a way that the subtraction

```
_updateLiquidity(-int256(_shareProceeds - _shareReservesDelta))
```

in HyperdriveLong.sol _applyCloseLong() will revert.
Exploit scenario / Numerical Example: Coded POC and POC's output: https://hackmd.io/@certora/HklsnIFYn.
Implications: Positions won't be closeable (funds are locked and unredeemable) as long as Lido's `TotalPooledEther` doesn't increase to at least its previous amount.
Delv's response: acknowledged and **fixed**: https://github.com/Delv/hyperdrive/pull/390.

Medium-06: Wrong accounting of share proceeds `_applyRemoveLiquidity()`

Severity: Medium
Probability: Low
Category: Logic/Calculation error
File(s): HyperdriveLP.sol
Bug description: The function `removeLiquidity()` in `HyperdriveLP.sol` calls `_applyRemoveLiquidity()` which in turn calls `_applyWithdrawalProceeds()`.
If the number of withdrawal shares calculated is negative, proceeds are updated by calculating an over-estimated value of the proceeds and then updating the liquidity by the delta compensation.
The share proceeds returned from this function are not updated, thus creating a discrepancy between the calculated withdrawal shares, the (re-) updated liquidity and the withdrawal share proceeds.
Implications: The result is thus both a discrepancy between the liquidity of the pool and an over-compensation of the withdrawal pool.
Delv's response: acknowledged and **fixed**:
https://github.com/Delv/hyperdrive/pull/376.

Medium-07: Discrepancy between share price calculations due to rounding errors

Severity: Medium
Probability: Low
Category: Logic/Calculation error
File(s): AaveHyperdrive.sol
Bug description: The function `_pricePerShare()` in the `AaveHyperdrive.sol` instance calculates the share price in a different way when depositing or withdrawing funds to the protocol. The different computation method leads to a discrepancy between the actual share price (assets/shares) and the one the checkpoint saves if it hasn't been set to any value yet.
Exploit scenario: For example, in `openLong()`, the share price is calculated by the output of `_deposit()`, but in `closeLong()`, it is used by the built-in `_pricePerShare()` function.
Note: The difference between these two calculations could get more severe, as the deposited amount decreases, making it a non-stable difference. Our error bound analysis shows that the share price for depositing `dx` amount, in a pool with assets `x0` and total shares `S` is bounded from above by: `Share price <= x0 * 1e18 / ( S - (x0-1)/dx )`.
Numerical Example:

```
Unset
S = totalShares = (10^20 - 1) / (10^5 - 1) = 1000010000100001
dx = amount = 99999
x0 = assets = 1e18
dS = newShares = 99
dS = floor(S * dx / x0)

Maximum error:
dS = (S * dx - x0 + 1)/x0
dS*x0 = S*dx - x0 + 1
(dS+1)*x0 = S*dx + 1
dS = 10^m - 1; m is an integer (m = 2)
x0 = 10^18

10^(m+18) = S*dx + 1
S*dx = 10^(m + 18) - 1
dx = [ 10^(m + 18) - 1 ] / S

dx = 10^5 - 1 = 99999
S = (10^20 - 1)/(10^5 - 1)

pricePerShare = floor(x0*1e18/S) = 999.99 (float point)
sharePrice = amount / newShares = floor(1e18*(10^5 - 1)/(10^2-1)) =
1010.0909… (float point)
```

Delv's response: acknowledged and **fixed**:
https://github.com/Delv/hyperdrive/pull/301.

Medium-08: Accuracy loss in average time calculations

Severity: Low
Probability: High
Category: Precision loss
File(s): HyperdriveLP.sol
Bug description: The inputs calculated in `HyperdriveLP.sol` for the average time have a significant loss of accuracy since there is a transition between 18 decimals to seconds and then back to 18 decimals. An error of 1e18 seconds in `longAverageMaturityTime` is then multiplied by a factor of `1e18/ _positionDuration` ~ 1e10.
Delv's response: acknowledged and **fixed**: https://github.com/Delv/hyperdrive/pull/304.

Medium-09: Hyperdrive: Denial-of-Service when `bondReserves` is zero

Severity: High
Probability: Low
Category: DOS
File(s): HyperdriveLong.sol
Bug description: Depending on the initial parameters of the contract, it's possible to open a long position that would drive the curve to zero-bonds point (`_marketState.bondReserves=0`). On such occurrence, no long/short position could be opened or closed, and no further liquidity could be provided. After the initial liquidity is provided at construction, the amount of assets needed to drive the system into the DoS state is:

$$\Delta z = z_0 \left(\left(1 + \frac{\mu d}{c} \right)^{\frac{1}{1-t}} - 1 \right)$$

Where `z0` is the initially supplied assets, and the other parameters parallel the ones that appear in the "YieldSpace with Yield Bearing Vaults" document.
Delv's response: acknowledged and **fixed**: https://github.com/Delv/hyperdrive/pull/304.

Low-01: Wrong Permit Typehash - Non-Compliance with EIP712

Severity: Low
Probability: 100%
Category: Programming error
File(s): MultiToken.sol
Bug description: There's a missing closing parenthesis in MultiToken.sol `PERMIT_TYPEHASH`, which changes the expected hash.
Implications: As it's still possible to sign off-chain, the only impact here would be visual (UI).
Delv's response: acknowledged and **fixed**: https://github.com/Delv/hyperdrive/pull/242.

Low-02: No fee bound

Severity: Low
Probability: Low
Category: Lack of boundaries for fees / Rug vector
File(s): HyperdriveFactory.sol: `constructor` and `updateFees()`
Bug description: There is no upper bound on the fee percentage as set in the configurator. That's why it can be at or over 100% (maliciously or mistakenly). Find more information, including POC here.
Exploit scenario: Governance can monitor the mempool for a big trade and sandwich attack by frontrunning and setting fees to 100%, then letting the trade happen, then back running and setting fees back to their initial value.
Implications: Trust issue.
Delv's response: acknowledged and **fixed**: https://github.com/Delv/hyperdrive/pull/241.

Low-03: `pricePerShare()` drop below initial value

Severity: Low
Probability: Low
Category: Logic/Calculation error
File(s): AaveHyperdrive.sol
Bug description: `_pricePerShare()` for `AaveHyperdrive.sol` can drop below its initial value, when a significant long is closed so that only some dust remains.
Exploit scenario / **Numerical Example**:
In a scenario when we have
`balanceOf(this) == totalShares == 2 * 10^27`
closing a long position is so big, that it almost depletes the pool. There can be a situation where : `balanceOf(this) == 8`, `totalShares == 9`.
This would mean that the `_pricePerShare()` drops from `10^18` to `10^18 * 8/9`.
Implications: wrong accounting that can result in protocol losses.
Delv's response: *We found an issue where calculateCloseLong was handling a decrease in pricePerShare incorrectly. Should be resolved with this PR* https://github.com/Delv/hyperdrive/pull/390.

Low-04: Consider using the existing `SafeCast` library's `toUint128()` and adding a `toUint224()` to prevent unexpected overflows when casting from various type `int`/`uint` values

Severity: Low
Category: Coding mispractice
File(s): HyperdriveLP.sol, HyperdriveTWAP.sol
Bug description: More information, including POC, is in the following link.
Delv's response: *The overflowing of the TWAP is fine and intentional (we use the same premise as Uniswap V2), and we've added safe casting here:* https://github.com/Delv/hyperdrive/pull/418.

Low-05: Return values of `transfer()`/`transferFrom()` are not checked

Severity: Low
Category: Coding mispractice
File(s): AaveHyperdrive.sol
Bug description: While ATokens seem to revert indeed when there's a failure in `transfer()`/`transferFrom()`, the function signature still has a `boolean` return value that indicates that everything went well. To avoid any potential silent failure, it's best to check that `success == true`.
More information, including POC, is in the following link.
Delv's response: *This is addressed by the removal of `AaveHyperdrive` in favor of the `ERC4626Hyperdrive` in combination with YieldDaddy's Aave wrapper:* https://github.com/Delv/hyperdrive/pull/454.

Low-06: `updateWeightedAverage()` math bounds exceeded

Severity: Low
Category: Input validation
File(s): FixedPointMath.sol
Bug description: The function output is not bounded by the average and the delta. A weighted average method should produce a value that is bounded between the old average and the delta.
Delv's response: acknowledged.
Revision: The output value of the function doesn't satisfy the exact mathematical bound due to rounding-errors. The actual calculated average doesn't exceed the theoretical new average:

$$average = (totalWeight * average \pm deltaWeight * delta) / (totalWeight \pm deltaWeight)$$

yet can deviate from it (towards zero) up to:

$$theoretical\ average - calculated\ average <= 1 + 2 * \frac{10^{18}-1}{totalWeight \pm deltaWeight}.$$

So the maximal possible deviation is when $totalWeight \pm deltaWeight = 1$:

$$theoretical\ average - calculated\ average <= 1 + 2 * (10^{18} - 1)$$

which was proven by the Certora prover.

Given that the average is an average of 18-decimal timestamp, we assert that the error is insignificant.

Fix: fixed in commit 78dee14.

Informational-01: Gas optimizations

Severity: Informational
Bug description: Certora's team has prepared a gas optimization report that can be found here.
Delv's response: *All Informational and Optimization Issues should be addressed in*
https://github.com/Delv/hyperdrive/commit/014108ab931ae8e2ea90a7db59e309144364977c
https://github.com/Delv/hyperdrive/commit/eba85fe66d4ef23681df70bec06b9bf94250a55d
Although we've opted to not include some optimizations due to the infamous issue of `stack to deep`*, and we will be moving away from via-ir due to a lack of trust in its process. Additionally, AaveHyperdrive and DsrHyperdrive we recently removed in favor of a re-consolidation around ERC4626 and so those have been skipped. As for the removal of initialization values in loop this does not affect gas, and so initialization value shave been left in for readability. Lastly, we feel the existing order of functions preserves readability over the Solidity standard. (ERC20Permit was also removed entirely in favor of solmate's ERC20 impl which is more efficient and battle-tested)*

Informational-02-24: Missing best practices

Severity: Informational
Bug description: Certora's team has prepared the report with missing best practices that weren't implemented in the Hyperdrive system. It can be found here.
Delv's response: the same as above.

Informational-25: Flagging lack of Check-Effect Interaction-Pattern involving state variables updates

Severity: Low
Category: Coding mispractice / vulnerable code
File(s): DsrHyperdrive.sol, AaveHyperdrive.sol, BondWrapper.sol, AaveHyperdriveFactory.sol, HyperdriveFactory.sol
Bug description: More information, including POC, is in the following link.
Implications: The external functions mentioned should use the `nonReentrant` modifier, and all external or public functions using the vulnerable internal ones should also use the modifier.
Delv's response: acknowledged and **fixed**:
https://github.com/Delv/hyperdrive/pull/455.

Formal Verification Process

Since most contracts are combined into a complete Hyperdrive system and can't be considered separately, we will present properties for Hyperdrive (except `HyperdriveTAWP.sol`) without splitting them into subsections. We also present properties for `BondWrapper.sol` and `AssetId.sol`.
The structure of properties:

1. <notation> <property description> (<*property name in spec code*>)
    - <property specific assumptions> (don't mix up with general assumptions)

Function names (and signatures) shall be written in Source code Pro font size 11 with the grey highlight, e.g., `foo(uint256)`

Notations

✅ Indicates the rule is formally verified.
❌ Indicates the rule is violated.
⏳ Indicates the rule is timing out.

Hyperdrive system properties

Assumptions

- Loop unrolling: We assume any loop can have at most 1 iteration.
- Some variables are fixed to specific values which represent the expected system state.
- Many functions from `FixedPointMath.sol` and `HyperdriveMath.sol` were substituted with the CVL2 code. The following functions were affected:
    - `updateWeightedAverage()`
    - `pow()`
    - `exp()`
    - `ln()`
    - `mulDivDown()`
    - `mulDivUp()`
    - `calculateBondsInGivenSharesOut()`
    - `calculateBondsOutGivenSharesIn()`
    - `calculateSharesInGivenBondsOut()`
    - `calculateSharesOutGivenBondsIn()`
    - `calculateBaseVolume()`
    - `calculateSpotPrice()`
    - `calculateAPRFromReserves()`
    - `calculateInitialBondReserves()`
    - `_calculatePresentValue()`
    - `calculateShortInterest()`

- Fees calculations in the function `_calculateFeesOutGivenBondsIn()` were ignored.
- Function `recordPrice()` from `HyperdriveTAWP.sol` was ignored.
- We often set the following attributes to constants in this table:

Variable	Constant
Aave pool index	2 RAYs
_timeStretch	45071688063194104
_checkpointDuration	86400
_positionDuration	31536000
_updateGap	1000
_marketState.shareReserves	10^18

- And these attributes we often bounded within an interval:

Variable	minimal value	maximal value
_marketState.bondReserves	10^18	max_uint256
_checkpoints.sharePrice	0	1000 × 10^18

Properties

1. ✅ Checks the balance difference after transferring aTokens (*aTokenTransferBalanceTest*).
    - Due to rounding errors of index division and multiplication, the difference is bounded by `index + 1/RAY`.
2. ✅ No action can change the share price for two different checkpoints at the same time. (*sharePriceChangesForOnlyOneCheckPoint*).
3. ✅ For every checkpoint, if the share price has been set, it cannot be reset (*cannotChangeCheckPointSharePriceTwice*).
4. ✅ No function, except `removeLiqudity()`, can change the total supply of two different tokens at the same time (*onlyOneTokenTotalSupplyChangesAtATime*).
5. ✅ Tokens could be minted at a time corresponding to position duration time in the future (since the latest checkpoint) (*mintingTokensOnlyAtMaturityTime*).

6. ✅ The total supply of tokens whose maturity time is later than the (maturity period + present time stamp) is always zero (*NoFutureTokens*).
7. ✅ The ready to redeem shares cannot exceed the total withdrawal shares (*WithdrawalSharesGEReadyShares*).
8. ✅ The sum of long tokens is greater or equal to the outstanding longs (*SumOfLongsGEOutstanding*).
    - For `closeLong()` function: if `_checkpoints[checkpointTime].sharePrice != 0`, then sum of long tokens also considers caller's already closed outstanding longs (there can be the case that `checkpointTime`'s longs are closed but tokens are not withdrawn).
9. ✅ The sum of short tokens is greater or equal to the outstanding shorts (*SumOfShortsGEOutstanding*).
    - For `closeShort()` function: if `_checkpoints[checkpointTime].sharePrice != 0`, then sum of short tokens also considers the caller's already closed outstanding shorts (there can be the case that `checkpointTime`'s shorts are closed but tokens are not withdrawn).
10. ✅ When calling `openLong()`, a long position is opened (*openLongReallyOpensLong*).
    - `_checkpoints[_checkpointTime].sharePrice` was set before.
11. ✅ Trader won't spend more baseTokens than maxDeposit when open short. (*dontSpendMore*).
12. ❌ Check the correctness of `updateWeightedAverageCheck()` function (*updateWeightedAverageCheck*). Catches the bug.
13. ❌ The share price cannot go below the initial price (*SharePriceAlwaysGreaterThanInitial*). Catches the bug.
14. ❌ `checkpoint()` function correctly sets `_checkpoints[time].sharePrice` (*checkPointPriceIsSetCorrectly*). Catches the bug.

BondWrapper.sol properties

Assumptions

- Loop unrolling: we assume any loop can have at most 3 iterations.
- Hyperdrive instance representing MultiToken was simplified by removing many parts of the code preserving only tokens flow. It can be found here.
- BondWrapper can't be a `msg.sender`.

Properties

1. ✅ `mint()` correctly updates msg.sender's MultiToken balance and destination's BondWrapper balance won't be decreased. (*mintIntegrityUser*).
    - Hyperdrive instance isn't a destination.
2. ✅ `mint()` correctly updates BondWrapper's MultiToken balance and BondWrapper's totalSupply won't be decreased. (*mintIntegritySystem*).
3. ✅ `mint()` doesn't affect the balances of other users on any token involved. (*mintIntegrityOthers*).
4. ✅ Splitting mint amount into two smaller mint amounts (amount1 and amount2) is not profitable. (*mintIntegritySmallsVsBig*).
    - amount1 + amount2 == amount.
5. ✅ `close()` correctly updates msg.sender's BondWrapper and destination's ERC20 balances. (*closeIntegrityUser*).
6. ✅ `close()` doesn't affect the balances of other users on any token involved. (*closeIntegrityOthers*).
7. ❌ A user can redeem minted tokens (*implementationCorrectness*). Catches the bug.

AssetId.sol properties

Properties

1. ✅ `encodeAssetId()` -> `decodeAssetId()` should return the same `_prefix` and `_timestamp` as were encoded initially. (*encodeDecodeInverse*).
2. ✅ `decodeAssetId()` -> `encodeAssetId()` should return the same `_id` as were decoded initially. (*decodeEncodeInverse*).