Curve Security Analysis Report and Formal Verification Properties

x

Table of Contents

Summary

This document describes the specification and verification of the **Curve Crypto Pools** using the Certora Prover and manual code review findings. The work was undertaken from **11th November 2023** to **4th February 2024**. The latest commit reviewed manually and run through the Certora Prover is [6d9d4f5487](6d9d4f5487).

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

curve-stablecoin Repo:

```
contracts/factory/OwnerProxy.vy
contracts/factory/StableswapFactory.vy
contracts/factory/StableswapFactoryHandler.vy
contracts/mpolicies/AggMonetaryPolicy.vy
contracts/mpolicies/AggMonetaryPolicy2.vy
contracts/price_oracles/AggregateStablePrice.vy
contracts/price_oracles/AggregateStablePrice2.vy
contracts/price_oracles/CryptoWithStablePrice.vy
contracts/price_oracles/CryptoWithStablePriceAndChainlink.vy
contracts/price_oracles/CryptoWithStablePriceAndChainlinkFrxeth.vy
contracts/price_oracles/CryptoWithStablePriceETH.vy
contracts/price_oracles/CryptoWithStablePriceFrxethN.vy
contracts/price_oracles/CryptoWithStablePriceTBTC.vy
contracts/price_oracles/CryptoWithStablePriceWBTC.vy
contracts/price_oracles/CryptoWithStablePriceWstethN.vy
contracts/price_oracles/EmaPriceOracle.vy
contracts/stabilizer/PegKeeper.vy
contracts/AMM.vy
contracts/Controller.vy
contracts/ControllerFactory.vy
contracts/Stablecoin.vy
```

The contracts are written in Vyper 0.3.9.

The Certora Prover demonstrated that the implementation of the Vyper 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 Vyper 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 acknowledged**	**Total fixed**
Critical	0		
High	2		
Medium	12		
Low	4		
Informational	5		
Total (High, Medium, Low)	**18**		

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: No Oracle Sanity Check

Severity: High
Probability: Medium
Category: Chainlink Oracle
File(s):
CryptoWithStablePriceAndChainlink.vy
CryptoWithStablePriceAndChainlinkFrxeth.vy
CryptoWithStablePriceETH.vy
CryptoWithStablePriceFrxethN.vy
CryptoWithStablePriceTBTC.vy
CryptoWithStablePriceWBTC.vy
CryptoWithStablePriceWstethN.vy

Any interaction with chain link price feeds should include a sanity check to ensure that latestRoundData() is non-zero. In our case, the check is missing. This seemingly should have been resolved by our use of Uniswap V3 TWAP Oracle to bound the min/max return values of the feed. However, the bounds prove ineffective if the feed returns zero.

For example, we can see that in CryptoWithStablePriceAndChainlinkFrxeth.vy (L197-L222):

```
197 @internal
198 @view
199 def _raw_price() -> uint256:
200     p_crypto_r: uint256 = TRICRYPTO.price_oracle(TRICRYPTO_IX)  # d_usdt/d_eth
201     p_stable_r: uint256 = STABLESWAP.price_oracle()              # d_usdt/d_st
202     p_stable_agg: uint256 = STABLESWAP_AGGREGATOR.price()       # d_usd/d_st
203     if IS_INVERSE:
204         p_stable_r = 10**36 / p_stable_r
205     crv_p: uint256 = p_crypto_r * p_stable_agg / p_stable_r     # d_usd/d_eth
206     price_per_share: uint256 = SFRXETH.pricePerShare()
207     p_staked: uint256 = min(STAKEDSWAP.price_oracle(), 10**18) * price_per_share / 10**18  # d_eth / d_sfrxeth
208 
209     chainlink_lrd: (uint80, int256, uint256, uint256, uint80) = CHAINLINK_AGGREGATOR.latestRoundData()
210     chainlink_p: uint256 = convert(chainlink_lrd[1], uint256) * 10**18 / CHAINLINK_PRICE_PRECISION
211 
212     lower: uint256 = chainlink_p * (100 - BOUND_SIZE) / 100
213     upper: uint256 = chainlink_p * (100 + BOUND_SIZE) / 100
214     crv_p = min(max(crv_p, lower), upper)
215 
216     crv_p = p_staked * crv_p / 10**18
217 
218     uni_price: uint256 = self._uni_price()
219     uni_price = min(uni_price * (100 - UNI_DEVIATION) / 100, chainlink_p) * price_per_share / 10**18
220     crv_p = max(crv_p, uni_price)
221 
222     return crv_p
```

If chainlink_lrd[1] = 0 then chainlink_p = lower = upper = 0 and then crv_p =

min(max(0,0),0)=0 in L214 and 216, so uni_price=min(...,0)*price_per_share/10**18 = 0, and finally the return value crv_p = max(0,0)=0 in L220.

Curve's response:

High-02: Attacker can sandwich Oracle updates to take under-collateralized loans

Severity: High
Probability: High
Category: Logic
File(s):
CryptoWithStablePriceAndChainlink.vy
CryptoWithStablePriceAndChainlinkFrxeth.vy
CryptoWithStablePriceTBTC.vy
CryptoWithStablePriceWBTC.vy
CryptoWithStablePriceWstethN.vy

Bug description: For an example of such a bug, see issue 5.1 in Bancor V2 Consensys Audit. This is a complicated problem to solve but there are a few ways to mitigate or get around this MEV problem in CDP protocols (including e.g. circuit breakers). A nice summary of existing approaches can be found in Liquidity's blog "The Oracle Conundrum", see also Angle's research series and Synthetix's blog in that regard for discussion.

Curve's response:

Medium-01: Missing Check for L2-Sequencer State

Severity: Medium
Probability: Low
Category: Chainlink Oracle, Optimistic Rollups
File(s):
CryptoWithStablePriceAndChainlink.vy
CryptoWithStablePriceAndChainlinkFrxeth.vy
CryptoWithStablePriceETH.vy
CryptoWithStablePriceFrxethN.vy
CryptoWithStablePriceTBTC.vy
CryptoWithStablePriceWBTC.vy
CryptoWithStablePriceWstethN.vy

Consider you have deployed a lending protocol on L2, and its sequencer goes down. This has happened in the past and may happen in the future. When the sequencer comes back online and oracles update their prices, all price movements that occurred during downtime are applied at once. If these movements are significant, they may cause chaos. Borrowers would rush to save their positions, while liquidators would rush to liquidate borrowers. Since liquidations are handled mainly by bots, borrowers are likely to suffer mass liquidations. This is unfair to borrowers, as they could not act on their positions even if they wanted to due to the L2 downtime.

Remark: note that curve.fi is deployed on eight L2 chains (including Arbitrum and Avalanche etc) at the time of writing this document. We should mention that e*ven if an L2 goes offline, anyone can still submit a transaction for L2 through L1* by something called a delayed inbox (arbitrum) OR canonical transaction chain (optimism). These transactions are always executed first when the sequencer comes back online. In theory, some borrowers can still avoid liquidation by closing their position through this delayed inbox. However, it is unlikely that normal borrowers would have the required knowledge to do so, creating unfair grounds for the same set of users. For a similar bug, see issue M-2 in Sherlock's Dodo audit.
Curve's response:

Medium-02: Not Checking For Stale Prices

Severity: High
Probability: Low
Category: Chainlink Oracle
File(s):
CryptoWithStablePriceAndChainlink.vy
CryptoWithStablePriceAndChainlinkFrxeth.vy

Bug description: In both files, we are receiving the chainlink price feed using latestRoundData,

```
chainlink_lrd: (uint80, int256, uint256, uint256, uint80) = CHAINLINK_AGGREGATOR.latestRoundData()
chainlink_p: uint256 = convert(chainlink_lrd[1], uint256) * 10**18 / CHAINLINK_PRICE_PRECISION
```

but there is no check if the return value indicates stale data. According to the Chainlink documentation, this could lead to stale prices, see [1] and [2].
Note that currently, Chainlink Oracles are out of use by a DAO decision.
Curve's response:

Medium-03: Delayed Inbox Attack

Severity: Medium
Probability: Low
Category: Optimistic Rollups
File(s):
CryptoWithStablePriceAndChainlink.vy
CryptoWithStablePriceAndChainlinkFrxeth.vy
CryptoWithStablePriceETH.vy
CryptoWithStablePriceFrxethN.vy
CryptoWithStablePriceTBTC.vy
CryptoWithStablePriceWBTC.vy
CryptoWithStablePriceWstethN.vy

This is essentially the "opposite problem" from the L2-Sequencer attack which stems from the lack-of-checks bug above. If the previous bug dealt with ordinary users and could be mitigated by knowledge of the delayed inbox, this scenario deals with what a technically savvy user could do by exploiting it to take undercollateralized loans.
Remark: this is a known L2 attack, a good explanation can be found here.
Curve's response:

Medium-04: Unhandled Oracle Reverts

Severity: High
Probability: Low
Category: Chainlink Oracle
File(s):
CryptoWithStablePriceAndChainlink.vy
CryptoWithStablePriceAndChainlinkFrxeth.vy
CryptoWithStablePriceETH.vy
CryptoWithStablePriceFrxethN.vy
CryptoWithStablePriceTBTC.vy
CryptoWithStablePriceWBTC.vy
CryptoWithStablePriceWstethN.vy

Remark: note that even though vyper does not support "try/catch" exception handling like solidity, we can mimic the same functionality using raw_call (see the excerpt from the documentation below) with the revert_on_failure flag set to False.

raw_call(*to: address, data: Bytes, max_outsize: uint256 = 0, gas: uint256 = gasLeft, value: uint256 = 0, is_delegate_call: bool = False, is_static_call: bool = False, revert_on_failure: bool = True*) → Bytes[max_outsize]

Call to the specified Ethereum address.

- `to`: Destination address to call to
- `data`: Data to send to the destination address
- `max_outsize`: Maximum length of the bytes array returned from the call. If the returned call data exceeds this length, only this number of bytes is returned. (Optional, default `0`)
- `gas`: The amount of gas to attach to the call. (Optional, defaults to `msg.gas`).
- `value`: The wei value to send to the address (Optional, default `0`)
- `is_delegate_call`: If `True`, the call will be sent as `DELEGATECALL` (Optional, default `False`)
- `is_static_call`: If `True`, the call will be sent as `STATICCALL` (Optional, default `False`)
- `revert_on_failure`: If `True`, the call will revert on a failure, otherwise `success` will be returned (Optional, default `True`)

Note: Returns the data returned by the call as a `Bytes` list, with `max_outsize` as the max length. The actual size of the returned data may be less than `max_outsize`. You can use `len` to obtain the actual size.

Returns nothing if `max_outsize` is omitted or set to `0`.

Returns `success` in a tuple with return value if `revert_on_failure` is set to `False`.

Curve's response:

Medium-05: Inverse of EMA is not EMA of inverses

Severity: Medium
Probability: High
Category: TWAP
File(s): CryptoWithStablePrice.vy

Bug description: In CryptoWithStablePrice.vy#L140-L146,

```
@internal
@view
def _raw_price() -> uint256:
    p_crypto_r: uint256 = TRICRYPTO.price_oracle(TRICRYPTO_IX)  # d_usdt/d_eth
    p_stable_r: uint256 = STABLESWAP.price_oracle()             # d_usdt/d_st
    p_stable_agg: uint256 = STABLESWAP_AGGREGATOR.price()       # d_usd/d_st
    if IS_INVERSE:
        p_stable_r = 10**36 / p_stable_r
    return p_crypto_r * p_stable_agg / p_stable_r
```

P_stable_r is inverted if the flag IS_INVERSE is True. This is fine for Uniswap V3 type TWAP's but wrong for other kinds (e.g., Uniswap V2). Looking at the implementation of the stable swap pool in Stableswap.vy#L484-L504, we find that the TWAP is computed using an exponential moving average

```
@internal
@view
def _ma_price() -> uint256:
    ma_last_time: uint256 = self.ma_last_time

    pp: uint256 = self.last_prices_packed
    last_price: uint256 = min(pp & (2**128 - 1), 2 * 10**18)
    last_ema_price: uint256 = shift(pp, -128)

    if ma_last_time < block.timestamp:
        alpha: uint256 = self.exp(- convert((block.timestamp - ma_last_time) * 10**18 / self.ma_exp_time, int256))
        return (last_price * (10**18 - alpha) + last_ema_price * alpha) / 10**18

    else:
        return last_ema_price


@external
@view
def price_oracle() -> uint256:
    return self._ma_price()
```

Therefore the computation is incorrect and would lead to loss of accuracy in the pricing information (thankfully, since the coins in question are stable, the price fluctuations would be small in general; otherwise this would be a critical error).

Remark: note that Oracle prices of token pairs (used for borrowing) *do not have to be symmetric* as opposed to spot prices (in which trade occurs).
Curve's response:

Medium-06: There is no limit on the fees users have to pay

Severity: Medium
Probability: Medium
Category: Logic, Input Validation
File(s): AMM.vy
Bug description:
We need to make sure that users' funds cannot be lost even when the protocol (or its admin) makes a mistake or acts maliciously. However, there is no upper limit on the percentage of fees users have to pay (i.e., the fee can be 100%!), and no upper limit on the percentage of fees that go to the admin. See lines 1683-L1692 in AMM.vy:

```
1683    @external
1684    @nonreentrant('lock')
1685    def set_fee(fee: uint256):
1686        """
1687        @notice Set AMM fee
1688        @param fee Fee where 1e18 == 100%
1689        """
1690        assert msg.sender == self.admin
1691        self.fee = fee
1692        log SetFee(fee)
```

As well as lines 1695-1764:

```
1695    @external
1696    @nonreentrant('lock')
1697    def set_admin_fee(fee: uint256):
1698        """
1699        @notice Set admin fee - fraction of the AMM fee to go to admin
1700        @param fee Admin fee where 1e18 == 100%
1701        """
1702        assert msg.sender == self.admin
1703        self.admin_fee = fee
1704        log SetAdminFee(fee)
```

Curve's response:

Medium-07: Integration with upgradable/proxy tokens is not done in a safe way

Severity: Medium
Probability: Medium
Category: Unusual tokens
File(s):
Bug description: Key stable coins (e.g. USDC, USDT) are upgradable, while others (e.g., tUSD) interact via a proxy. All of them are traded in curve pools on Mainnet:

USDC

ALL 90 | USD | BTC | ETH | CRVUSD | TRICRYPTO | CRYPTO

Sort by Volume (desc) | Hide very small pools

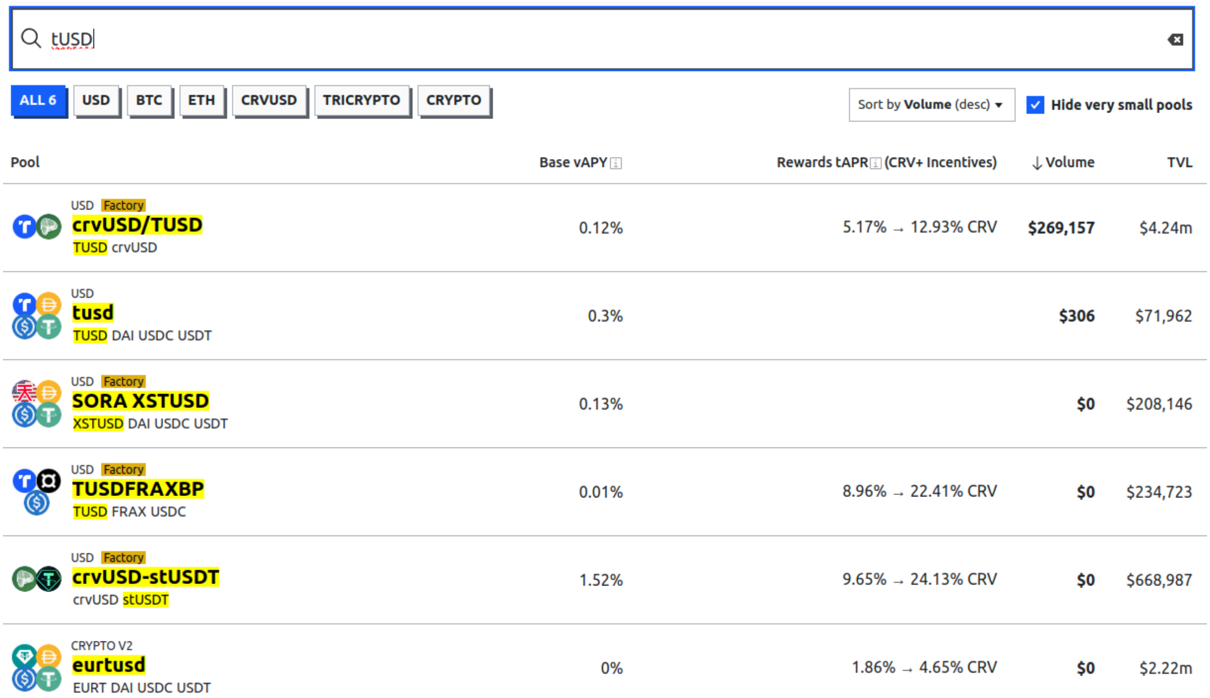
Pool	Base vAPY	Rewards tAPR (CRV+ Incentives)	↓ Volume	TVL
USD 3pool DAI USDC USDT	0.96%	0.96% → 2.41% CRV	$100.01m	$194.40m
USD Factory crvUSD/USDC USDC crvUSD	0.56%	3.81% → 9.53% CRV	$10.97m	$35.81m
USD susd DAI USDC USDT sUSD	0.54%	1.27% → 3.17% CRV	$2.44m	$16.72m
USD fraxusdc FRAX USDC	0.05%	2.75% → 6.87% CRV	$1.70m	$136.32m
CRYPTO V2 Factory TricryptoUSDC USDC WBTC ETH	0.27%	5.79% → 14.47% CRV	$1.63m	$38.95m
USD lusd LUSD DAI USDC USDT	0.67%	0.080% → 0.20% CRV	$1.13m	$17.99m
USD Factory Prisma mkUSD mkUSD FRAX USDC	0.4%	0.53% → 1.35% CRV	$341,487	$30.08m

USDT

ALL 47 | USD | BTC | ETH | CRVUSD | TRICRYPTO | CRYPTO

Sort by Volume (desc) | Hide very small pools

Pool	Base vAPY	Rewards tAPR (CRV+ Incentives)	↓ Volume	TVL
USD 3pool DAI USDC USDT	0.96%	0.96% → 2.41% CRV	$100.01m	$194.40m
CRYPTO V2 tricrypto2 USDT WBTC ETH	1.06%	1.83% → 4.57% CRV	$17.30m	$74.81m
USD Factory crvUSD/USDT USDT crvUSD	0.32%	4.94% → 12.35% CRV	$5.51m	$31.36m
USD susd DAI USDC USDT sUSD	0.54%	1.27% → 3.17% CRV	$2.44m	$16.72m
USD lusd LUSD DAI USDC USDT	0.67%	0.080% → 0.20% CRV	$1.13m	$17.99m
CRYPTO V2 Factory TricryptoUSDT USDT WBTC ETH	0.34%	4.83% → 12.09% CRV	$476,066	$47.17m
CRYPTO V2 Factory MMX/USDT MMX USDT	4.63%		$293,978	$1.89m



A change to the token semantics can break any smart contract that depends on past behavior. Thus, developers integrating with such ERC20 tokens should freeze the pool in which they are traded until the change has been reviewed and approved by governance. An example of such an adapter can be found in MakerDAO's GemLike6 interface.

Curve's response:

Medium-08: Bridged assets de-pegging is unaccounted for

Severity: Medium
Probability: Medium
Category: Chainlink Oracle, bridges
File(s):
CryptoWithStablePriceWBTC.vy,
ape-wbtc-oracle.py,
CryptoWithStablePriceTBTC.vy,
ape-tbtc-oracle.vy.

Bug description: In the Curve stablecoin system, users can deposit tBTC/wBTC as collateral and borrow crvUSD against it. The bug here is partially in the deployment script and partially in the contract logic.

To understand it, examine the code for the relevant method used in the wBTC pool which computes the prices (the situation for tBTC is similar):

```
def _raw_price(tvls: uint256[N_POOLS], agg_price: uint256) -> uint256:
    weighted_price: uint256 = 0
    weights: uint256 = 0
    for i in range(N_POOLS):
        p_crypto_r: uint256 = TRICRYPTO[i].price_oracle(TRICRYPTO_IX[i])   # d_usdt/d_btc
        p_stable_r: uint256 = STABLESWAP[i].price_oracle()                 # d_usdt/d_st
        p_stable_agg: uint256 = agg_price                                  # d_usd/d_st
        if IS_INVERSE[i]:
            p_stable_r = 10**36 / p_stable_r
        weight: uint256 = tvls[i]
        # Prices are already EMA but weights - not so much
        weights += weight
        weighted_price += p_crypto_r * p_stable_agg / p_stable_r * weight     # d_usd/d_btc
    crv_p: uint256 = weighted_price / weights

    # Limit BTC price
    if self.use_chainlink:
        chainlink_lrd: ChainlinkAnswer = CHAINLINK_AGGREGATOR_BTC.latestRoundData()
        if block.timestamp - min(chainlink_lrd.updated_at, block.timestamp) <= CHAINLINK_STALE_THRESHOLD:
            chainlink_p: uint256 = convert(chainlink_lrd.answer, uint256) * 10**18 / CHAINLINK_PRICE_PRECISION_BTC
            lower: uint256 = chainlink_p * (10**18 - BOUND_SIZE) / 10**18
            upper: uint256 = chainlink_p * (10**18 + BOUND_SIZE) / 10**18
            crv_p = min(max(crv_p, lower), upper)

    return crv_p
```

And note that in line 21 of the deployment script (ape-wbtc-oracle.py) of the wBTC pool, the oracle address used is:

```
21      CHAINLINK_BTC = "0xf4030086522a5beea4988f8ca5b36dbc97bee88c"
```

But unlike tri-crypto, Stableswap, and stable aggregator price oracles used in the code and deployed to addresses

```
STABLECOIN = "0xf939E0A03FB07F59A73314E73794Be0E57ac1b4E"
```

```
TRICRYPTO = ["0x7F86Bf177Dd4F3494b841a37e810A34dD56c829B", "0xf5f5B97624542D72A9E06f04804Bf81baA15e2B4"]  # USDC, USDT
```

```
AGG = "0x18672b1b0c623a30089A280Ed9256379fb0E4E62"
```

Which all measure the following *on-chain* (hereby "on-chain" we mean - accessible on the Ethereum Mainnet) quantities and are fed into the formula

```
        weighted_price += p_crypto_r * p_stable_agg / p_stable_r * weight     # d_usd/d_btc
    crv_p: uint256 = weighted_price / weights
```

The chainlink oracle address used is not the wBTC/USD price but rather the *off-chain* BTC/USD price. Thus, there is a hidden assumption here:

1 WBTC on Ethereum = 1 BTC on Bitcoin

This is not necessarily true in a situation where the wBTC bridge is compromised (or went offline) but both chains continued to trade as usual. In such a case, it is likely that wBTC would temporarily depeg from BTC, but the protocol will continue to price wBTC using the BTC/USD price, even though wBTC will instantly become worth far less than native BTC due to the bridge compromise. Users could then buy wBTC for a far lower value than native BTC, deposit it into the protocol, and borrow against it using the value of native BTC, systematically taking an under-collateralized loan. Worse still, since crvUSD is a cross-collateralization protocol, the damage will not be confined to the specific wrapped asset pool but would allow the attackers to undermine the stablecoin itself in the event of a bridge compromise leading to a depeg event.

Remark: apriori, since the chainlink price is only supposed to safeguard the limits of aggregated TVL-weighted price, it would seem that the value of crv_p (see code above, L165-178) would protect the users from harm. However, note that if the lower boundary of chainlink's answer is higher than crv_p (which can happen during a depegging event), the data from the on-chain price oracles is ignored, regardless of how much TVL they have.
Note that currently, Chainlink Oracles are out of use by a DAO decision.

Recommendations: In immediate terms, the wBTC pool is live and cannot be upgraded or frozen, however, the chainlink oracle in it can and has been turned off (for unrelated reasons). If governance chooses to turn it on for any reason, they should take this issue under advisement. Moving forward and to help address this issue for future wrapped assets, if the protocol chooses to use Chain link (or some other form of off-chain data feed) it is advised that the protocol would use Chainlink's wBTC/BTC price feed (or its equivalent product) to monitor for a depeg event of a wrapped asset.

Curve's response:

Medium-9: Calculation misses BORROWED_PRECISION

Severity: Medium
Probability: High
Category: computation (unit conversion)
File(s): AMM.vy
Bug description: In `withdraw` sometimes the dust is added to admin_fees, when the last share is withdrawn. However, this computation misses the scale: it does not divide by BORROWED_PRECISION. This overestimates the admin fees and admins can accidentally or intentionally withdraw more stable coins than they should.
See Integrity of withdraw property (AMM section)

Curve's response:

Medium-10: No "remove_market" in the Controller Factory

Severity: High
Probability: Low
Category: logic
File(s): ControllerFactory.vy
Bug description: The Controller Factory contains the method add_market which adds a new market (i.e., Controller/AMM pair) from a blueprint. The address of the blueprint can be found in the global parameters controller_implementation and amm_implementation which can be changed by the admin via set_implementations. However, in case a mistake was made there is no way for the admin to later remove it.

Curve's response:

Medium-11: the contract "CurveStableSwap2Balances" does not include a manipulation guard and should not be used for maintaining the Peg

Severity: Medium
Probability: Medium
Category: logic, TWAP
File(s): ControllerFactory.vy
Bug description: There is a variant of the stable swap pool called "CurveStableSwap2Balances" which is used to support positive-rebasing and fee-on-transfer tokens. However, unlike the standard stable swap pool, this

variant does not include the recent changes incorporated into the code to protect against TWAP manipulation. Compare (stableswap):

```
@internal
@view
def _ma_price() -> uint256:
    ma_last_time: uint256 = self.ma_last_time

    pp: uint256 = self.last_prices_packed
    last_price: uint256 = min(pp & (2**128 - 1), 2 * 10**18)
    last_ema_price: uint256 = shift(pp, -128)
```

With (CurveStableSwap2Balances):

```
@internal
@view
def _ma_price() -> uint256:
    ma_last_time: uint256 = self.ma_last_time

    pp: uint256 = self.last_prices_packed
    last_price: uint256 = pp & (2**128 - 1)
    last_ema_price: uint256 = pp >> 128
```

In particular, this is the case for the live implementation of USDM (an ERC20 stablecoin issued by Mountain Protocol and backed by T-bills) which was considered as a candidate for PegKeeping in crvUSD.

Curve's response:

Medium-12: exchange_dy can send back less than the requested output amount

Severity: High
Probability: Medium
File(s): AMM.vy
Bug description: The function `exchange_dy` is supposed to protect the user from bad trades by checking the `max_in_amount`, but it doesn't check the output amount. The function `calc_swap_in` can return a partial trade that does not cover the whole output amount if one of several conditions occurs: too many bands iterated, the last band reached, or the price exceeds a limit. In that case, it returns a trade with a smaller `amount_out` than requested. Such a trade is then executed as long as the input amount is less than the maximum amount. This buys tokens at a much higher price or sells them for a lower price than requested by the user.
The problem is a missing check that the user gets the requested amount, together with the feature in the `calc_swap_in` function that can compute trades that send less than the requested amount.

While it's unlikely that a user hurts himself accidentally, this can be exploited for example by sandwich attacks that buy to empty all but the highest price band, then execute the user's `exchange_dy` transactions at a much higher price than requested and then revert their previous trade with a profit.

Recommendation: explain this potential pitfall to smart contract wallets or protocols that integrate with the AMM.

See Integrity of exchange_dy property (AMM section)

Curve's response:

Low-01: Setup has no input range checks

Severity: Low
Probability: Low
Category: Initialisation
File(s): AMM.vy
Bug description: The constructor does not check if the values for some critical variables are in range. E.g. COLLATERAL_PRECISION and BORROWED_PRECISION can be zero, which will lead to a strange behavior, e.g., in `withdraw`. This is exacerbated because this function uses `unsafe_div` on these values.

Curve's response:

Low-02: Mishandled Oracle price decimals

Severity: Medium
Probability: Low
Category: Chainlink Oracle
File(s):
CryptoWithStablePriceAndChainlink.vy
CryptoWithStablePriceAndChainlinkFrxeth.vy

The code that converts the raw data of chainlink's price feed (stored in the variable chainlink_lrd) into the form we use in the project (stored in the variable chainlink_p)
is correctly designed to deal with accuracy loss due to division by first multiplying the original value by 10^18:

```
172        chainlink_lrd: (uint80, int256, uint256, uint256, uint80) = CHAINLINK_AGGREGATOR.latestRoundData()
173        chainlink_p: uint256 = convert(chainlink_lrd[1], uint256) * 10**18 / CHAINLINK_PRICE_PRECISION
```

But there is an assumption here: the maximal number of decimals returned by chainlink's price feed is 18. However, while price feeds *usually* return 8 or 18

decimals for token pairs, there are exceptions (even on the Ethereum mainnet). An example of such a token is the NEAR token which has 24 decimals, located at address:
0x85f17cf997934a597031b2e18a9ab6ebd4b9f6a4,
and whose price feed can be observed at the address:
0xC12A6d1D827e23318266Ef16Ba6F397F2F91dA9b
In such cases, loss of accuracy would occur. For a parallel discussion of such a bug, see the comments in issue M-6 in Sherlock's Dodo audit.

Curve's response:

Low-03: An attacker can take a loan at the expense of another user

Severity: High

Probability: Low

Category: Logic

File(s): Controller.vy

Bug description: This is a composability issue - consider the external function create_loan_extended appearing in lines 620-644:

```
620 @payable
621 @external
622 @nonreentrant('lock')
623 def create_loan_extended(collateral: uint256, debt: uint256, N: uint256, callbacker: address, callback_args: DynArray[uint256,5]):
624     """
625     @notice Create loan but pass stablecoin to a callback first so that it can build leverage
626     @param collateral Amount of collateral to use
627     @param debt Stablecoin debt to take
628     @param N Number of bands to deposit into (to do autoliquidation-deliquidation),
629            can be from MIN_TICKS to MAX_TICKS
630     @param callbacker Address of the callback contract
631     @param callback_args Extra arguments for the callback (up to 5) such as min_amount etc
632     """
633     # Before callback
634     STABLECOIN.transfer(callbacker, debt)
635 
636     # Callback
637     # If there is any unused debt, callbacker can send it to the user
638     more_collateral: uint256 = self.execute_callback(
639         callbacker, CALLBACK_DEPOSIT, msg.sender, 0, collateral, debt, callback_args).collateral
640 
641     # After callback
642     self._create_loan(0, collateral + more_collateral, debt, N, False)
643     self._deposit_collateral(collateral, msg.value)
644     assert COLLATERAL_TOKEN.transferFrom(callbacker, AMM.address, more_collateral, default_return_value=True)
```

which calls the internal function execute_callback appearing in lines 546-571:

```
546    @internal
547    def execute_callback(callbacker: address, callback_sig: bytes4,
548                         user: address, stablecoins: uint256, collateral: uint256, debt: uint256,
549                         callback_args: DynArray[uint256, 5]) -> CallbackData:
550        assert callbacker != COLLATERAL_TOKEN.address
551
552        data: CallbackData = empty(CallbackData)
553        data.active_band = AMM.active_band()
554        band_x: uint256 = AMM.bands_x(data.active_band)
555        band_y: uint256 = AMM.bands_y(data.active_band)
556
557        # Callback
558        response: Bytes[64] = raw_call(
559            callbacker,
560            concat(callback_sig, _abi_encode(user, stablecoins, collateral, debt, callback_args)),
561            max_outsize=64
562        )
563        data.stablecoins = convert(slice(response, 0, 32), uint256)
564        data.collateral = convert(slice(response, 32, 32), uint256)
565
566        # Checks after callback
567        assert data.active_band == AMM.active_band()
568        assert band_x == AMM.bands_x(data.active_band)
569        assert band_y == AMM.bands_y(data.active_band)
570
571        return data
```

For the vulnerability to occur, one must have a Smart Contract Wallet (or Protocol) in some address A which approves the Controller contact for its tokens and which has a fallback function that returns a non-zero value. In such a case, an attacker in address B can call this function with callbacker=address A with some collateral and debt constants and have the response of the callbacker interpreted as agreement to issuing more_collateral.

Recommendation: explain this potential pitfall to smart contract wallets or protocols that integrate with the Controller.

Curve's response:

Low-04: An admin can manipulate the EMA oracle of a Stableswap pool to its benefit

Severity: High
Probability: Low
Category: TWAP, Input Validation
File(s): Stableswap.vy
Bug description: the function `set_ma_exp` (in lines 1097-1102) allows an admin to set ma_exp_time to *any positive uint256 value*, including 1.

```
1097    @external
1098    def set_ma_exp_time(_ma_exp_time: uint256):
1099        assert msg.sender == Factory(self.factory).admin()  # dev: only owner
1100        assert _ma_exp_time != 0
1101
1102        self.ma_exp_time = _ma_exp_time
```

In such a case, the value of alpha in _ma_price() becomes very small and the moving average can be reduced to essentially the spot price of the stablecoin, which itself can be manipulated to be up to two dollars (*twice* the expected price). This is already bad by itself. But we further note that such a sharp price change is also quite likely to influence the stable Aggregator, triggering liquidations and threatening the stability of the crvUSD CDP.

Remark: note that even if the admin is fully trusted, any concentration of power in the hands of a single wallet (or multi-sig) creates a centralization risk with the associated web2 and cryptographic risks. For example, many DeFi projects (e.g., the Wintermute attack) were hacked due to the vulnerability of the Profanity address generator in the Vanity wallet.
Note that currently, Chainlink Oracles are out of use by a DAO decision.

```
@internal
@view
def _ma_price() -> uint256:
    ma_last_time: uint256 = self.ma_last_time

    pp: uint256 = self.last_prices_packed
    last_price: uint256 = min(pp & (2**128 - 1), 2 * 10**18)
    last_ema_price: uint256 = shift(pp, -128)

    if ma_last_time < block.timestamp:
        alpha: uint256 = self.exp(- convert((block.timestamp - ma_last_time) * 10**18 / self.ma_exp_time, int256))
        return (last_price * (10**18 - alpha) + last_ema_price * alpha) / 10**18

    else:
        return last_ema_price


@external
@view
def price_oracle() -> uint256:
    return self._ma_price()


@internal
def save_p_from_price(last_price: uint256):
    """
    Saves current price and its EMA
    """
    if last_price != 0:
        self.last_prices_packed = self.pack_prices(last_price, self._ma_price())
        if self.ma_last_time < block.timestamp:
            self.ma_last_time = block.timestamp


@internal
def save_p(xp: uint256[N_COINS], amp: uint256, D: uint256):
    """
    Saves current price and its EMA
    """
    self.save_p_from_price(self._get_p(xp, amp, D))
```

Curve’s response:

Informational-01: No protection against signature malleability

Severity: Medium
Probability: Low
Category: Cryptography
File(s): Stablecoin.vy
Bug description:
As is well-known, ECDSA signatures are naturally malleable (i.e., non-unique) and can be modified while maintaining validity. Unfortunately, the EVM precompile `ecrecover` does not include protection against such an attack. Thus it is best practice to require that the `s` value of any valid signature would be in the lower half order, and require the `v` value to be either 27 or 28. See OpenZeppelin ECDSA implementation and the inline comments there.

Informational-02: Certain pathological tokens (e.g., XAUt) are incompatible with the pool

Severity: Low
Probability: Low
Category: Unusual tokens
File(s):
Bug description: Some tokens do not return bool (e.g. USDT, OMG) on ERC20 methods. see here for a comprehensive (if somewhat outdated…) list. Worse still some tokens (e.g. BNB) may return a bool for some methods, but fail to do so for others. These cases are handled in the code of crvUSD in a similar way to OpenZeppelin's SafeTransfer abstraction, i.e., by adding an assert and setting the default return values to true. However, some are pathological (but not necessarily marginal!) ERC20 tokens declare a boolean return value but may return false even when the transfer is successful. An example of such a token is Tether Gold whose current fully diluted valuation is ~$500,000,000.

Recommendations: do not whitelist such tokens, or if necessary write a special adapter for them.

Informational-03: Use of a deprecated bit shift operation

Severity: Informational
Probability: Low
Category: Use of a deprecated function
File(s): multiple contracts in the project

Bug description: The function shift was deprecated in Vyper 0.3.8 (see e.g. comment in https://docs.vyperlang.org/en/latest/built-in-functions.html). It should probably be replaced by << (or >>) since the declared Vyper version is 0.3.9.

Informational-04: Clash of default value

Severity: Informational
Probability: Low
Category: Best-practice
File(s): OwnerProxy.vy, StableSwap.vy, ape_deploy.py

The default _ma_exp_time parameter in OwnerProxy is set to 600 but it is set to 866 (=600/ln(2)) in StableSwap initialization code and also when deploying via the ape_deploy script it is changed to 866.

Informational-05:Integration with fee-on-transfer/deflationary tokens can cause the pool to be exploited

Severity: High
Probability: Low
Category: Unusual tokens
File(s): many places
Bug description: crvUSD carries out its own internal accounting strictly based on the assumption that when `transfer` and `transferFrom` succeed they transfer the precisely correct amount. For example, consider the exchange code below (the vulnerable lines are marked with yellow):

```
    in_amount_done: uint256 = unsafe_div(out.in_amount, in_precision)
    out_amount_done: uint256 = unsafe_div(out.out_amount, out_precision)
    if use_in_amount:
        assert out_amount_done >= minmax_amount, "Slippage"
    else:
        assert in_amount_done <= minmax_amount, "Slippage"
    if out_amount_done == 0 or in_amount_done == 0:
        return [0, 0]

    out.admin_fee = unsafe_div(out.admin_fee, in_precision)
    if i == 0:
        self.admin_fees_x += out.admin_fee
    else:
        self.admin_fees_y += out.admin_fee

    n: int256 = min(out.n1, out.n2)
    n_start: int256 = n
    n_diff: int256 = abs(unsafe_sub(out.n2, out.n1))

    for k in range(MAX_TICKS):
        x: uint256 = 0
        y: uint256 = 0
        if i == 0:
            x = out.ticks_in[k]
            if n == out.n2:
                y = out.last_tick_j
        else:
            y = out.ticks_in[unsafe_sub(n_diff, k)]
            if n == out.n2:
                x = out.last_tick_j
        self.bands_x[n] = x
        self.bands_y[n] = y
        if lm.address != empty(address):
            s: uint256 = 0
            if y > 0:
                s = unsafe_div(y * 10**18, self.total_shares[n])
            collateral_shares.append(s)
        if k == n_diff:
            break
        n = unsafe_add(n, 1)

    self.active_band = out.n2

    log TokenExchange(_for, i, in_amount_done, j, out_amount_done)

    if lm.address != empty(address):
        lm.callback_collateral_shares(n_start, collateral_shares)

    assert in_coin.transferFrom(msg.sender, self, in_amount_done, default_return_value=True)
    assert out_coin.transfer(_for, out_amount_done, default_return_value=True)
```

However, some important tokens take a transfer fee (e.g. PAXG with current FDV of ~467,000,000$), some do not currently charge a fee but may do so in the future (e.g. USDT, USDC - see the previous bug regarding upgradability). This behavior has led to exploits in the past, e.g., the STA transfer fee was used in a sophisticated attack to drain $500,000 from several Balancer pools ([more details]()).

Formal Verification Process

The structure of properties:

1. <notation> <property description> (<*property name in spec code*>)
    - <property specific assumptions>

Notations

✅Indicates the rule is formally verified.
❌Indicates the rule is violated.

<u>Assumptions</u>

- Loop unrolling: We assume any loop can have at most 3 iterations. For some of the rules, we had to further reduce the number of loop iterations because the loop body was too complex.
- Some variables are fixed to specific values which represent the expected system state.
- `log2()` function and other price-related functions were substituted with the CVL2 code that abstracts from concrete values.
- Mocks and CVL summaries were used for contracts like collateral, WETH, and Factory.

Controller properties

1. ✅ Integrity Of create_loan -
    - The user didn't have a loan before
    - The correct loan is added to the loan array under e.msg.sender
    - Loans[length -1] == msg.sender
    - loans_ix[msg. Sender] == loans.length -1
    - Minted_before + debt = minted after
    - Deposited collateral balances are updated correctly
    - User's stablecoin balance is updated correctly
2. ✅ Integrity Of add_collateral -
    - Revert if loan does not exist
    - Debts do not change
    - Collateral is moved from msg.sender to AMM
3. ✅ Integrity Of remove_collateral -
    - Revert if loan does not exist
    - Debts do not change
    - Collateral is moved from AMM to msg.sender

4. ✅ Integrity Of borrow_more -
    - Minted amount is increased by “debt” added
    - Collateral balance updated correctly (sender dec & contract inc)
    - StableCoin balances are updated correctly (sender inc & contract dec)
    - User’s loan is updated correctly
5. ✅ borrow_more is cumulative - calling borrow_more twice is equivalent to calling borrow_more one time with the sum of parameters called two times.
6. ✅ Integrity Of repay -
    - “_for” debt is decreased correctly.
    - The repaid amount is sent from msg.sender to the controller.

    if the debt is repaid fully:
    - collateral is transferred from AMM to "_for" loan.
    - already changed stable coins are transferred from AMM and used to decrease debt.
7. ✅ Integrity of repay_extended -
    - debt of msg.sender is decreased.

    If partially repaid:
    - collateral is moved only between AMM and callbacker (sum stays the same)
    - AMM withdraw returned no stablecoins.
    - callbacker paid debt: callbacker balance decreases bythe difference of debt before and after.
    - e.msg.sender stablecoin and collateral balances are unchanged.

    Otherwise:
    - debtAfter is 0.
    - sum of the stablecoin balance of callbacker and sender after equals balance before minus debtBefore plus withdrawn stable.
    - sum of collateral balance of callbacker and sender afterwards equals balance before minus withdrawn collateral.
    - balance of AMM decreases by withdrawn stable/withdrawn collateral.

8. ✅ repay is cumulative - calling repay twice is equivalent to calling repay one time with the sum of parameters called two times.
    - We only show the equivalence of balance changes.
    - On the AMM shares the user can actually lose some shares due to rounding when repay is called twice. This is because the assets are repositioned in the AMM into different bands and because of rounding errors in the shares.
9. ✅ Can liquidate only if health is less than zero.
    - We do not check here under which economic conditions health is negative.
    - A user can always liquidate themselves, even if healthy.
10. ✅ User can always borrow more when the condition suits it.
11. ✅ A user can't have more than one active loan per system instance.
12. ✅ Only liquidate can decrease other user's AMM shares.
13. ✅ Any position can be closed.
14. ✅ If a user has a loan then he has shares.
15. ✅ Changing liquidation discount doesn't affect existing loans
16. ✅ Invariant that the loans and loans_ix are inverse of each other.
    - Either idx is 0 and lent amount is 0,
    - or idx is smaller than n_loans and loans[idx]=user.
    - For every index idx between 0 and n_loans-1, we have loans_ix[loans[idx]]=idx.
17. ✅ AMM gets its funds - The Controller is responsible for sending and withdrawing funds to the AMM:
    - When funds = withdraw() is called, the controller must withdraw the funds given by the return value funds[0] (for stable) and funds[1] (for collateral)
    - When reset_admin_fees() is called, the controller must withdraw the value of active_fees_x/active_fees_y (at the beginning of this call; they are reset to 0 by the call)
    - When deposit_bands() is called, the controller must deposit the given amount of collateral to the AMM.

- We checked that the controller updates the AMM balance accordingly. We assume that the AMM never calls functions of the controller (like taking a loan) and is not the fee receiver.
- In the AMM we checked that withdraw(), deposit_bands() and remove_admin_fees() change the sumof[bands]+admin_fees by the same amount. For exchange and exchange_dy we checked that both sumof[bands]+admin_fees and the token balance change by the same amount. For all other functions in the AMM, we showed that they don't affect token balance nor sumof[bands].
- In summary, this shows that the token balance is always at least the amount sumof[bands]+admin_fees.

AMM properties

1. ✅ Integrity of deposit_range -
    - sumof(bands_y) + admin_fees increases by amount*COLLATERAL_PRECISION
    - sumof(bands_x) + admin_fees stays the same.
2. ❌ Integrity of withdraw -
    - User_shares after = user_shares before * frac/1e18
    - Total_x, total_y = assets of removed user_shares
    - Sumof[bands_x] + admin_fees decreases by (at least) total_x * BORROWED_PRECISION

      This part of the rule is broken, **see Medium-9**. We proved it only under the assumption that BORROWED_PRECISION == 1.
    - Sumof[bands_y] + admin_fees decreases by (at least) total_y * COLLATERAL_PRECISION
    - Share/asset ratio should only go up and stay roughly the same.
3. ✅Integrity of exchange -
    - Collateral/borrowed balance change = change of sumof[bands_x/y]
    - balance(msg.sender) of in token decreases by at most in_amout
    - balance(this) of in token increases by the same amount.
    - balance(_for) of out token increases by at least min_amount.
    - balance(this) decreases by the same amount.
4. ❌Integrity of exchange_dy

- Collateral/borrowed balance change = change of sumof[bands_x/y]
- balance(msg.sender) of in token decreases by at most max_amount.
- balance(this) of in token increases by the same amount.
- balance(_for) of out token increases by at least out_amount.
  This assertion is broken for exchange_dy, **see Medium-12.**
- balance(this) of out token decreases by the same amount.

5. ✅ Only active_band may have both types of tokens - Bands below active_band() are completely sold (y=0), Bands above active_band() are completely unsold (x=0).
6. ✅ Unpack invariant for user_share ticks - user shares are stored packed in the ticks array with two uint128 packed in one uint256. Furthermore, if the first element is 0 all user_shares are 0, regardless of the content of this array. Also, elements outside the n1 to n2 range are 0 regardless of what the ticks array contains.
7. ✅ total_shares equals the sum of user_shares - The desired property is
   total_shares[n] == SUM_n user_tick_unpack[u][n]
   where the unpacked user_tick is computed according to the unpack invariant.
8. ✅ The Stablecoin balance of the AMM Contract exceeds admin fees plus the sum of x in all bands - (BORROWED_TOKEN.balance(this) - admin_fee_y) * BORROWED_PRECISION >= sum_of(bands_y).
   - In the AMM we proved that this holds for all functions except withdraw, deposit_range and reset_admin_fees().
   - In the Controller we proved (17.) that whenever one of the three functions is called, the corresponding tokens are transferred
   - We assume that the token is not fee-taking or deflationary (Medium-09)
9. ✅ Collateral balance of the AMM Contract exceeds the value in bands and admin fee - (COLLATERAL_TOKEN.balance(this) - admin_fee_x) * COLLATERAL_PRECISION >= sum_of(bands_x).
   - In the AMM we proved that this holds for all functions except withdraw, deposit_range and reset_admin_fees().

- In the Controller we proved (17.) that whenever one of the three functions is called, the corresponding tokens are transferred
- We assume that the token is not fee-taking or deflationary (Medium-09)

10. ✅ Exchange does not change user's shares - Checks that user shares do not change after the exchange function is called.
11. ✅ Can never deposit in a band that is already partially or totally sold.
12. ✅ Ratios "total_shares[n] / collateral_balance" and "total_shares / stablecoin" balance should only increase on withdraw/deposit_range functions and only minimally due to rounding. We show this for an arbitrary band n. Split into three rules:
    - ✅ The ratio total_shares to bands_x in withdraw
        - the ratio (bands_x + 1) to (totalShares + DEAD_SHARES) only increases.
        - it doesn't increase if we would issue one more asset to the withdrawer.
        - If the band is completely emptied (no shares left), the ratio changes to the default ratio (1 to DEAD_SHARES).
    - ✅ The ratio total_shares to bands_y in withdraw
        - the ratio (bands_y + 1) to (totalShares + DEAD_SHARES) only increases.
        - it wouldn't increase if we would issue one more asset to the withdrawer.
        - If the band is completely emptied (no shares left), the ratio changes to the default ratio (1 to DEAD_SHARES).
    - ✅ The ratio total_shares / bands_y in DepositRange
        - the ratio (bands_y + 1) to (totalShares + DEAD_SHARES) only increases (increase happens due to rounding).
        - it wouldn't increase if we would issue one more share to the depositor.
13. ✅ Bands below min band are empty
    - n < min_band => bands_x(n) == 0 && bands_y(n) == 0
14. ✅ Bands above max_band are empty

- n > max_band => bands_x(n) == 0 && bands_y(n) == 0

15. ✅ Withdrawing when not in soft liquidation
    - when not in soft liquidation, withdraw will only withdraw collateral, not borrowed token: active_band_with_skip() < user_n1(user) => withdraw(user, frac)[0] == 0.
16. ✅ get_rate_mul equals set_rate result
    - get_rate_mul() should return the same value as set_rate(new_rate). The new_rate is only used afterwards and the current rate_mul is returned.
    - Also the next call to get_rate_mul() returns the same value, as long as all calls happen at the same block.timestamp.
17. ✅ deposit/withdraw removes or creates shares as appropriate
    - If the deposit succeeds, the user had no shares before and he has shares afterward.
    - If withdraw with frac=1 succeeds, the user had shares before and has no shares after.
    - If withdraw with frac!=1 succeeds, the user had shares before and still has after.

Stablecoin properties

1. ✅ The sum of all user's balances equals the total supply (*SumAllBalancesEqTotalSupply*).
2. ✅ Balance of address 0 is always 0 (*ZeroAddressNoBalance*).
3. ✅ Verify that there is no fee on transferFrom() (*noFeeOnTransferFrom*).
4. ✅ Verify that there is no fee on transfer() (*noFeeOnTransfer*).
5. ✅ Token transfer() works correctly. Balances are updated if returns true. Else, transfer amount was too high, or the recipient is 0. (*transferCorrect*).
6. ✅ Token transferFrom() works correctly. Balances are updated if returns true. Otherwise, transfer amount was too high, or the recipient is 0. (*transferFromCorrect*).

7. ✅ transferFrom should revert if and only if the amount is too high (higher than balance or allowance) or the recipient is 0 (*transferFromReverts*).
8. ✅ Contract calls don't change the token total supply. Except minting and burning functions ().
9. ✅ Transfer from a to b using transfer doesn't change the balance of other addresses (*TransferDoesntChangeOtherBalances*).
10. ✅ Transfer from a to b using transferFrom doesn't change the balance of other addresses (*TransferFromDoesntChangeOtherBalances*).
11. ✅ Allowance changes correctly as a result of calls to approve, approveAndCall, transferFrom, and permit (*ChangingAllowance*).
12. ✅ Verify that mint works correctly. Balances and totalSupply are updated correctly according to the parameters (*integrityOfMint*).
13. ✅ Verify that burn works correctly. Balances and totalSupply are updated correctly according to the parameters (*integrityOfBurn*).
14. ✅ Verify that burnFrom works correctly. Balances and totalSupply are updated correctly according to the parameters (*integrityOfBurnFrom*).
15. ✅ Verify that mint is monotonic (*mintMonotonicity*).
16. ✅ Verify that burn is monotonic (*burnMonotonicity*).

StableSwap properties

1. ✅ transferFrom changes the balances and allowances correctly (*transferFromChangesBalanceAndAllowanceCorrectly*).
2. ✅ transfer changes the balances correctly (*transferChangesBalanceCorrectly*).
3. ✅ transferFrom reverts iff the allowance is low, the balance is low or there is an overflow (balance + transferred sum > max_uint) (*transferFromRevertingConditions*).
4. ✅ approve sets allowance of intended parties as expected (*approveSetsAllowance*).
5. ✅ approve does not revert (*approveDoesNotRevert*)
6. ✅ transfer reverts iff the funds of the sender are low and if the recipient's balance overflows. (*transferRevertingConditions*).

7. ✅ Total supply can be changed only by add_liquidity or remove_liquidity (and its variants) (*totalSupplyDoesNotChange*).
8. ✅ No function can change balance of anyone who is not passed as an argument or is not a msg.sender (*doesNotAffectAThirdPartyBalance*)
9. ✅ No function can change allowance of anyone who is not passed as a parameter or is not a msg.sender. Function permit is not checked (*doesNotAffectAThirdPartyAllowance*).
10. ✅ Only transfer, transferFrom and add_liquidity can increase balance. Only transfer, transferFrom and remove_liquidity can decrease balance (*onlyAllowedMethodsMayChangeBalance*).
11. ✅ transfer can change balances of only msg.sender and receiver, transferFrom of sender and receiver, add/remove_liquidity only of msg.sender or intended recipient if specified (*whoCanChangeBalance*).
12. ✅ Only allow, permit can increase allowance. Only allow, permit and transferFrom can decrease allowance (*onlyAllowedMethodsMayChangeAllowance*).
13. ✅ Only add_liquidity can increase total supply. Only remove_liquidity (all 3 variants) can decrease total supply (*onlyAllowedMethodsMayChangeTotalSupply*).