CHAINSECURITY

PUBLIC

Code Assessment
of the Aggregate Stable Price V4 Smart Contracts

November 24, 2026

Produced for

Curve



Contents

1	Executive Summary	3
2	Assessment Overview	4
3	System Overview	5
4	Trust Model	8
5	System Considerations	10
6	Terminology	11
7	Findings	12
8	Limitations and use of report	23

1 Executive Summary

Dear Curve Team,

Thank you for trusting us to help Curve with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Aggregate Stable Price V4 according to **Scope** to support you in forming an opinion on their security risks.

Curve implements a stablecoin price oracle that combines prices from multiple Curve StableSwap pools. The aggregator assigns capped liquidity weights and reduces the influence of prices that deviate from the weighted reference price. It supports both legacy and StableSwap-NG pools and exposes the resulting price to downstream protocols.

The most critical subjects covered in our audit are price correctness and availability, and the resistance of liquidity-based weighting to manipulation. The most severe finding allowed a zero-supply legacy pool to block oracle reads and was corrected in **Version 2**, see **A Zero-Supply Legacy Pool Blocks the Aggregator**.

Temporary liquidity can still inflate a legacy source's stored TVL through permissionless checkpointing, a risk accepted by Curve, see **Temporary Liquidity Can Increase a Legacy Source's Weight**. Inflation at registration was corrected in **Version 2** by starting new legacy sources with zero stored TVL; since **Version 3**, registration also takes a checkpoint so that the phase-in starts at registration, see **Initial Legacy TVL Can Be Inflated**.

The general subjects covered are error handling, administrative configuration, code maintainability, event handling, and gas efficiency. Version 2 corrected the pool-registration issues and improved source validation. Version 3 made source removal independent of source indices, bounded the TVL averaging window, and prevented extreme liquidity floors from blocking price calculations. Safe operation depends on trusted administration selecting independent sources and suitable parameters.

The reviewed codebase provides a high level of security following the corrections in **Version 2** and **Version 3**, provided proper administration and parameter selection. All remaining findings are accepted or acknowledged risks.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,
ChainSecurity

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Aggregate Stable Price V4 repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

Version	Date	Commit Hash	Note
1	28 August 2026	6961e39468fa9e8386d6337de8317ecae7c6478e	Initial review
2	31 October 2026	2fc553de1d99d260b4f6d4881c48ca4a79e5ce13	Fixes
3	24 November 2026	8a5c8bca010b3105dab8b4d32841843ab867c5eb	Fixes

For the Vyper smart contracts, the compiler version `0.4.3` was chosen.

```
curve_stablecoin/
└── price_oracles/
    └── aggregate_stable_price/
        ├── AggregateStablePrice4.vy
        ├── capped_share.vy
        └── weighted_price.vy
```

2.1.1 Excluded from scope

Anything not listed above is considered out of scope for this assessment. This includes, but is not limited to, other Vyper smart contracts, tests, scripts, and documentation files.

3 System Overview

This system overview describes the most recent version (**Version 3**) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Curve's Aggregate Stable Price V4 system calculates a USD-denominated price for a configured stablecoin from the oracle prices and liquidity of multiple Curve StableSwap pools. The main `AggregateStablePrice4` contract incorporates the `capped_share` and `weighted_price` modules. Together, the three scoped files select sufficiently liquid sources, assign capped liquidity weights, penalize sources whose prices deviate from the group, and expose the resulting price with 18 decimal places.

3.1 Price Sources

The admin can register up to 20 distinct StableSwap pools. A registered pool must contain the configured stablecoin as one of its first two coins.

The aggregator supports both legacy and StableSwap-NG pools. It classifies a pool as NG if a static call to `price_oracle(uint256)` succeeds and returns at least 32 bytes; otherwise, it treats the pool as a legacy pool. It subsequently reads prices through `price_oracle(0)` for NG pools and `price_oracle()` for legacy pools. If the configured stablecoin is coin 0, the reported pool price is inverted as `10^36 / price` so that all source prices use the same 18-decimal orientation. Registration checks that the resulting price is nonzero and exercises the applicable liquidity read path.

3.2 Liquidity Measurement and Source Selection

Pool liquidity determines both whether a source participates and its base aggregation weight. The aggregator obtains liquidity differently for the two supported pool types:

- For an NG pool, it uses the pool's `D_oracle()` value directly. Any smoothing of this value is therefore provided by the pool.
- For a legacy pool, it derives the current TVL as `totalSupply() * get_virtual_price() / 10^18` and smooths it against the last stored TVL with an exponential moving average. If the LP-token supply is zero, the current TVL observation is zero without calling `get_virtual_price()`. The moving-average time constant, `tvl_ma_time`, defaults to 50,000 seconds and can be set by the admin to any positive value up to 30 days.

For elapsed time `dt`, the legacy-pool update uses `alpha = exp(-dt / tvl_ma_time)` and calculates `current_tvl * (1 - alpha) + stored_tvl * alpha`. Registration first checkpoints the existing sources, so a legacy pool starts with zero stored TVL and phases in through the EMA from the time of registration. Because an NG pool counts with its full `D_oracle()` value immediately, new legacy sources are underweighted relative to NG sources until they have phased in over several multiples of `tvl_ma_time`. Only pools whose resulting TVL is at least `min_liquidity` are active sources, so a new legacy source remains inactive until its stored TVL has grown to this threshold. This threshold defaults to 100,000 units with 18 decimals and can be set to any positive value by the admin. If no source is active, the aggregator returns the price stored by the last checkpoint, `last_price`. This value is initialized to `10^18`, corresponding to 1 USD, and remains unchanged until a source becomes active again.

3.3 Capped Liquidity Weights

The `capped_share` module converts the active sources' TVLs into relative weights. It uses a water-filling algorithm: any source whose proportional share exceeds the applicable cap is fixed at that cap, after which the remaining share is redistributed among uncapped sources in proportion to their TVLs. This limits the direct weight assigned to any single liquidity source.

Unless the admin configures a custom cap, the cap depends on the number of active sources:

Active Sources	Maximum Weight per Source
0 or 1	100%
2	70%
3 to 5	45%
6 or more	24%

The admin may set a custom cap between 1% and 100%. If the cap multiplied by the number of active sources is at most 100%, the base weights become effectively equal regardless of TVL. Setting it to zero restores the default schedule.

3.4 Price Aggregation

The `weighted_price` module first calculates a reference price as the weighted average of all active source prices using their capped TVL weights. It then calculates each source's squared price deviation from this reference, normalized by the admin-configurable `sigma` parameter, which must be between `10^9` and `10^18`, inclusive. A smaller value causes prices away from the reference to lose influence more quickly.

Each capped TVL weight is multiplied by an exponential penalty derived from this normalized deviation. The smallest deviation among sources with nonzero weight is subtracted from all deviations before exponentiation to improve numerical behavior without changing their relative penalties. The final oracle value is the weighted average of the source prices under these penalized weights. Consequently, a source's influence depends on both its capped relative liquidity and its agreement with the capped-liquidity reference price.

3.5 Price Reads and Checkpointing

The aggregator exposes two price paths:

- `price()` is a view function. It calculates the current effective TVLs and aggregate price without changing storage.
- `price_w()` is permissionless and state-changing. On the first call after `block.timestamp` advances, it checkpoints the effective TVLs, timestamp, and aggregate price. Further calls with the same timestamp return the stored price without querying or updating the sources again. `add_price_pair()` and `set_tvl_ma_time()` perform the same checkpoint before applying their change.

3.6 Administration and Emergency Source Removal

The admin controls the oracle configuration. It can add price sources, remove them by pool address, set or reset the custom share cap, set `sigma`, `min_liquidity`, and `tvl_ma_time`, replace the admin and emergency admin, and set the emergency admin's removal allowance. Admin removals are unrestricted by this allowance.

The emergency admin can only remove existing sources and must spend one unit of `emergency_remove_count` for each removal. It cannot add sources or change parameters.

3.7 Changes in Version 2

- The admin can now adjust `sigma`, the minimum liquidity threshold, and the legacy TVL EMA time constant.
- New legacy sources start with zero stored TVL and phase in through the EMA. A zero LP-token supply yields a zero current TVL observation without querying the virtual price.
- Source registration rejects duplicate pools, classifies a pool as NG only if the interface probe succeeds and returns at least 32 bytes, and checks the price and liquidity read paths.

3.8 Changes in Version 3

- Source registration and changes to `tvl_ma_time` first checkpoint the existing sources. `tvl_ma_time` is bounded by 30 days.
- Sources are removed by pool address instead of by index.
- The NG interface probe in `add_price_pair()` is now a static call.
- Zero-weight sources no longer affect the exponential penalty baseline.
- If no source is active, the aggregator returns the last checkpointed price instead of `10^18`.

4 Trust Model

The Aggregate Stable Price oracle is non-upgradeable and does not custody user assets. Its stablecoin address is fixed at deployment, while its set of price sources, source-weight cap, price-deviation parameter (`sigma`), minimum liquidity threshold, and legacy TVL EMA time constant remain under administrative control. Consequently, failures primarily affect systems that consume the reported price.

4.1 Roles

Admin

- Trust level: Fully trusted.
- Permissions: Add and remove price-source pools, set or reset the custom source share cap, set `sigma`, the minimum liquidity threshold, and the legacy TVL EMA time constant (up to 30 days), appoint a new admin or emergency admin, and set the emergency admin's removal allowance.
- Assumptions: The admin only configures compatible StableSwap pools that contain the intended stablecoin, chooses suitable values for the share cap, `sigma`, liquidity threshold, and TVL EMA time constant, and assigns the emergency role and its allowance securely. The admin also verifies that each configured pool is a legitimate, well-configured price source; the contract's checks alone do not establish this.
- Worst-case impact: A malicious or compromised admin can add adversarial price sources, remove legitimate sources, or select unsafe aggregation parameters and thereby arbitrarily distort the reported price or make oracle reads revert. It can also raise the liquidity threshold to exclude all sources, so that the oracle keeps returning the last checkpointed price, `last_price`. In the worst case, it can make the oracle return any malicious price of its choosing by configuring malicious price sources.

Emergency admin

- Trust level: Partially trusted.
- Permissions: Remove any configured price source, consuming one unit of the removal allowance set by the admin for each removal. It cannot add sources, change parameters, or manage roles.
- Assumptions: The emergency admin removes sources only in response to a faulty or compromised pool. The admin assigns an allowance appropriate to the number and diversity of configured sources.
- Worst-case impact: A malicious or compromised emergency admin can selectively remove up to its allowance of legitimate sources. This can reduce source diversity, cause the remaining sources to determine a manipulated price, and, if all sources are removed, cause the oracle to keep returning the last checkpointed price, `last_price`.

Permissionless callers

- Trust level: Untrusted.
- Permissions: Read the current price, effective share cap, and EMA TVLs. Any caller may invoke `price_w()` to checkpoint the current EMA TVLs and price; only the first successful checkpoint after the timestamp advances beyond `last_timestamp` updates storage. The admin functions `add_price_pair()` and `set_tvl_ma_time()` perform the same checkpoint.
- Assumptions: The system must remain safe regardless of who calls `price_w()` or when it is called. Integrators are responsible for calling the appropriate view or checkpointing function and interpreting its WAD-scaled return value.
- Worst-case impact: If pool state can be manipulated, an adversary may checkpoint that state into the TVL EMA and cache the corresponding price for the rest of the block.

4.2 External Dependencies

Configured StableSwap pools

The aggregator depends on every configured pool implementing the expected interface and returning valid values from `coins()`, `price_oracle()`, `price_oracle(uint256)`, `totalSupply()`, `get_virtual_price()`, and/or `D_oracle()`, according to the detected pool type. It assumes pool oracle prices and liquidity measurements are sufficiently manipulation-resistant and that the pools remain callable. A reverting or incompatible pool can make aggregate price reads revert. Incorrect or manipulated prices or TVLs can distort both the source weights and the reported aggregate price.

Interpreting the result as a USD price assumes that each counterpart asset is worth one USD.

Source diversity and market liquidity

Security relies on a sufficiently diverse set of independent pools meeting the configured minimum liquidity threshold (initially `100,000e18`). The share cap applies to base liquidity weights; the exponential penalty can subsequently concentrate influence on a source. In particular, capped sources receive equal base weights whatever their TVL, so when they disagree by several `sigma`, the smaller sources holding the remaining weight decide which price prevails. Correlated failures and price manipulations of pools can move the aggregate price. Specifically, the capping mechanism caps base liquidity weights and can improve source diversity; however, this can generally reduce the overall TVL needed to be manipulated in order to move the aggregate price. If no source meets the liquidity threshold, the oracle returns the last checkpointed price, `last_price` (initially `1e18`), which does not reflect the current market.

Oracle consumers

Downstream systems are expected to understand that the returned price is WAD-scaled, may be cached for a block when obtained through `price_w()`, and is the last checkpointed price, `last_price`, when no source is active. A manipulated or stale price can propagate to any protocol that relies on this oracle, potentially causing incorrect accounting, pricing, or liquidation decisions.

5 System Considerations

We leverage this section to highlight findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require a modification inside the project. Instead, they should raise awareness in order to improve the overall understanding for users and developers.

SC1 The Oracle Holds the Last Checkpointed Price with No Active Sources

System Consideration | Version 1

When no source meets the configured `min_liquidity`, AggregateStablePrice4 has no price to aggregate. Before **Version 3**, it returned `1e18` in this case, and a new `price_w()` checkpoint also stored this value in `last_price`.

In **Version 3**, `price()` and `price_w()` return `last_price` instead, and checkpoints leave it unchanged. This is the price stored by the last checkpoint taken while a source was active, or `1e18` if there has been none. It can therefore predate the moment the sources became inactive, differ from the last value returned by `price()`, and reflect a source that has since been removed. The oracle reports this price without a time limit until a source becomes active again.

SC2 Price Updates Are Checkpointed Once per Block

System Consideration | Version 1

`price()` recomputes the price on every call. The first `price_w()` call in a block checkpoints a price, which later `price_w()` calls in that block return unchanged.

This also applies when the admin changes parameters after a checkpoint. In **Version 3**, calls to `set_sigma()`, `set_min_liquidity()`, `set_share_cap()`, or `remove_price_pair()` can change the value returned by `price()`, while `price_w()` continues to return the cached `last_price` for the rest of the block.

`add_price_pair()` and `set_tvl_ma_time()` checkpoint the existing sources before applying the change. If no checkpoint exists yet in the block, they store a `last_price` computed with the previous configuration, which `price_w()` then returns for the rest of the block.

6 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- **Likelihood** represents the likelihood of a finding to be triggered or exploited in practice
- **Impact** specifies the technical and business-related consequences of a finding
- **Severity** is derived based on the likelihood and the impact

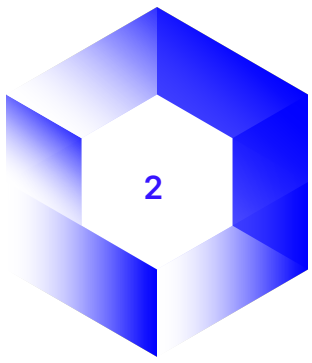
We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	**Impact**		
	High	Medium	Low
High	**Critical**	**High**	**Medium**
Medium	**High**	**Medium**	**Low**
Low	**Medium**	**Low**	**Low**

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

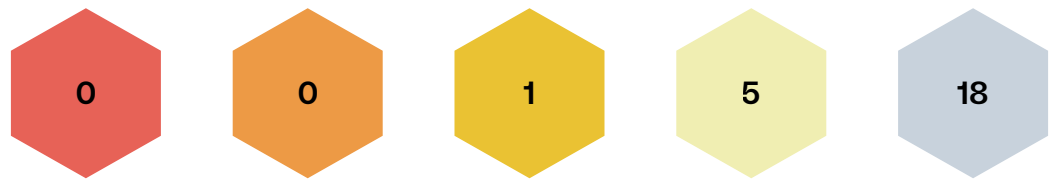
7 Findings

In this section, we describe the findings identified during the course of the engagement. The findings are categorized by severity as explained in the Terminology section. Below we provide an overview of the total number of findings per severity, as well as the number of open issues.



Open Issues

Total number of findings



Critical High Medium Low Informational

7.1 Overview

The following table lists all identified findings along with their severity and current status. A finding is considered resolved once it has been fully corrected or the specification has been changed accordingly. Non-informational findings with any other status, including partial fixes, acknowledgements, and accepted risks, remain open.

Medium	#001	A Zero-Supply Legacy Pool Blocks the Aggregator	Code Corrected
Low	#002	A Custom Share Cap Can Discard TVL Differences	Specification Changed
Low	#003	A Failed NG Probe Selects the Legacy Interface	Code Corrected
Low	#004	Initial Legacy TVL Can Be Inflated	Code Corrected
Low	#005	Temporary Liquidity Can Increase a Legacy Source's Weight	Risk Accepted
Low	#006	The Price Can Jump When a Source Crosses the Liquidity Floor	Risk Accepted
Info	#007	A Granted Emergency Removal Budget Cannot Be Reliably Revoked	Acknowledged
Info	#008	Emergency Removals Can Eliminate Source Diversity	Acknowledged
Info	#009	Events Declare No Indexed Fields	Code Corrected
Info	#010	Incompatible Pools Fail Only After Registration	Code Corrected
Info	#011	Legacy Sources Start with Inaccurate TVL	Code Corrected
Info	#012	Only the First Two Coins of an NG Pool Can Be Used	Specification Changed
Info	#013	Price-Pair Metadata Is Loaded Twice per Active Source	Acknowledged
Info	#014	Removed Sources Remain Visible Through Public Getters	Code Corrected
Info	#015	Reseeding an Empty NG Pool Reinitializes Its D Oracle	Acknowledged
Info	#016	Shared Constants Are Duplicated Across Modules	Acknowledged
Info	#017	The Same Pool Can Be Configured as Several Independent Price Sources	Code Corrected
Info	#018	A Very Low Liquidity Floor Can Block Price Calculations	Code Corrected
Info	#019	Immutable `SIGMA` Accepts a Wide, Partly Redundant Range	Code Corrected
Info	#020	Pending Removals Can Delete the Wrong Price Source	Code Corrected
Info	#021	Source Caps Can Give Smaller Pools a Combined Majority	Acknowledged

Info	#022	The Liquidity Floor and TVL Averaging Window Cannot Be Adjusted	Code Corrected	✓
Info	#023	TVL Observations Have No Relative Change Bound	Acknowledged	⊖
Info	#024	TVL Window Changes Can Abruptly Reweight or Freeze Legacy Sources	Code Corrected	✓

7.2 Descriptions

#001 A Zero-Supply Legacy Pool Blocks the Aggregator

Correctness | Medium | Version 1 | Code Corrected

When updating a legacy source's TVL, AggregateStablePrice4 calls `get_virtual_price()` before applying `MIN_LIQUIDITY`. Curve StableSwap calculates virtual price by dividing by LP-token supply, so the call reverts when the source has zero supply. From the next block, `price()`, `price_w()`, `ema_tvl()`, and `share_cap()` all revert until the pool receives liquidity or the source is removed.

Code Corrected

In **Version 2**, `_legacy_tvl()` returns zero when `totalSupply()` is zero without calling `get_virtual_price()`, allowing a drained legacy source's stored TVL to decay through the EMA without reverting aggregate reads.

#002 A Custom Share Cap Can Discard TVL Differences

Design | Low | Version 1 | Specification Changed

In `capped_share`, a custom cap is accepted without considering the number of active sources. If `n_active * custom_share_cap <= WAD`, the resulting base weights are effectively equal regardless of TVL, apart from integer rounding. The reference price is therefore effectively an unweighted mean, although the exponential penalty is still applied afterward.

The active count can later fall as source TVLs change or sources are removed. An existing custom cap can therefore enter this state without another `set_share_cap()` call.

Specification Changed

In **Version 2**, the NatSpec of `set_custom_share_cap()` and `capped_weights()` documents that a sufficiently low cap produces equal base weights regardless of TVL. This is an intended configuration, and selecting a cap consistent with the expected number of active sources is the admin's responsibility.

#003 A Failed NG Probe Selects the Legacy Interface

Correctness | Low | Version 1 | Code Corrected

In AggregateStablePrice4, `add_price_pair()` classifies a source as NG only when its low-level `price_oracle(uint256)` probe succeeds. The call uses `revert_on_failure=False`, so an unsupported selector, a revert, and an out-of-gas failure all return `False`. Every probe failure is interpreted as evidence that the source is legacy rather than causing registration to revert.

If the probe of an NG pool fails during registration, the pool is stored with `is_ng = False`. Once the source is active, `_get_p()` calls the zero-argument `price_oracle()`, which StableSwap-NG does not implement, so aggregate price reads revert.

Code Corrected

In **Version 2**, `add_price_pair()` classifies a pool as NG only if the probe succeeds and returns at least 32 bytes, and validates the selected interface through `_get_p()` before storing the source. An NG pool classified as legacy therefore fails registration when the zero-argument `price_oracle()` call reverts.

#004 Initial Legacy TVL Can Be Inflated

Security	Low	Version 1	Code Corrected

In AggregateStablePrice4, `add_price_pair()` initializes a legacy source's `last_tvl` from one `totalSupply()` read without EMA smoothing. A caller can invoke `price_w()`, temporarily add liquidity, permissionlessly execute a passed governance proposal that registers the pool, and withdraw in one transaction.

Since `last_timestamp` is then current, the inflated seed is initially used without dilution and can keep the source active or overweighted until the EMA decays. For NG sources, the seeded `last_tvl` is ignored on the next calculation because `_ema_tvl()` reads `D_oracle()` directly.

Code Corrected

In **Version 2**, `add_price_pair()` no longer seeds `last_tvl`, so legacy sources start with zero stored TVL.

As of **Version 3**, `add_price_pair()` checkpoints the existing sources before registration, so the phase-in of a new legacy source starts at registration. As a consequence, legacy sources are underweighted relative to NG sources for several multiples of `tvl_ma_time` after registration, so a newly configured aggregator needs this time before its price reflects all sources.

#005 Temporary Liquidity Can Increase a Legacy Source's Weight

Security	Low	Version 1	Risk Accepted

AggregateStablePrice4 updates a legacy pool's TVL EMA from its current `totalSupply() * get_virtual_price() / WAD` whenever permissionless `price_w()` creates a checkpoint. A caller can add liquidity, call `price_w()`, and remove the liquidity in one transaction, leaving part of the temporary TVL in `last_tvl`.

Unlike NG pools, whose liquidity oracle is maintained by pool interactions, legacy TVL observations are persisted only when the aggregator is checkpointed. Because the new observation receives more weight as the time since the previous checkpoint increases, a permissionless caller can effectively make the TVL at their chosen instant represent the entire elapsed period.

The stored value can then increase the source's weight, keep it above the liquidity floor, or change the default-cap tier until the EMA decays.

Risk Accepted

Curve accepts the inherited sampling model, citing frequent production checkpoints, per-source caps, and the additional cost of manipulating the pool's EMA price oracle as mitigating factors.

#006 The Price Can Jump When a Source Crosses the Liquidity Floor

Design	Low	Version 1	Risk Accepted

AggregateStablePrice4 excludes a source below the minimum liquidity threshold and includes it immediately when it reaches the floor. When no custom cap is set, crossing the 1-to-2, 2-to-3, or 5-to-6 active-source boundary also changes the cap from 100% to 70%, 70% to 45%, or 45% to 24%. If source prices differ, adding the source and reweighting the others can make the reported price jump.

The out-of-scope `AggMonetaryPolicy4` uses the reported price in its exponential rate calculation, so such price jumps can also cause borrowing rates to jump.

Risk Accepted

Curve accepts the price steps inherent to the inclusion threshold and tiered cap schedule, considering the smoothed TVLs and pool prices sufficient to limit their impact.

#007 A Granted Emergency Removal Budget Cannot Be Reliably Revoked

Informational	Version 1	Acknowledged

In AggregateStablePrice4, the emergency admin can spend its outstanding removal allowance before a pending admin transaction reduces or increases that allowance.

Acknowledged

Curve states:

> The emergency admin is a trusted role (a multisig); the budget bounds the damage of that role rather than defending against it. A race between spending and revoking an allowance is inherent to any allowance scheme. The admin can also rotate `emergency_admin` to the zero address in the same transaction as the allowance change.

#008 Emergency Removals Can Eliminate Source Diversity

Informational	Version 1	Acknowledged

The emergency admin may remove any sources until `emergency_remove_count` is exhausted. If it removes every source, subsequent price calculations return `1e18`; if it leaves only one source, that source alone determines newly calculated prices while active. Either state persists until the admin adds sources again, which is expected to require a slow governance process.

As of **Version 3**, the oracle returns the last checkpointed price instead of `1e18` when no source is active, see The Oracle Holds the Last Checkpointed Price with No Active Sources.

Acknowledged

Curve states:

> Removing sources is the purpose of the emergency role and `emergency_remove_count` is the tool that limits it. Governance keeps the count strictly below the number of configured sources. With no active source the oracle holds the last checkpointed price instead of reading a compromised source.

#009 Events Declare No Indexed Fields

Informational **Version 1** **Code Corrected**

AggregateStablePrice4 and `capped_share` declare no indexed event fields. An RPC log query can filter these logs by contract and event signature, but not by pool, admin, or pair index.

> **Code Corrected**
>
> In **Version 2**, `AddPricePair` and `RemovePricePair` index the pool address. `AddPricePair` also emits `is_ng`, and `RemovePricePair` includes the removed pool address.

#010 Incompatible Pools Fail Only After Registration

Informational **Version 1** **Code Corrected**

In AggregateStablePrice4, `add_price_pair()` classifies a source with a `price_oracle(uint256)` probe and reads both coins and `totalSupply()` before storing it.

- For a source classified as NG, registration does not call `D_oracle()`, and `price_oracle(uint256)`'s returned data size is not checked.
- For a source classified as legacy, it does not call the zero-argument `price_oracle()` or `get_virtual_price()`.

An incompatible Curve pool can therefore be stored successfully and later make aggregate price reads or liquidity-dependent getters revert when the aggregator first uses a method that registration did not call. `remove_price_pair()` makes no pool calls, so the source remains removable while aggregate reads revert.

> **Code Corrected**
>
> In **Version 2**, `add_price_pair()` classifies a pool as NG only if the probe succeeds and returns at least 32 bytes, and validates the selected price and TVL read paths before storing a source. It rejects zero prices and failed reads.

#011 Legacy Sources Start with Inaccurate TVL

Informational **Version 1** **Code Corrected**

In AggregateStablePrice4, `add_price_pair()` initializes `last_tvl` with `totalSupply()`, while `_ema_tvl()` measures a legacy pool's TVL as `totalSupply() * get_virtual_price() // WAD`. When the virtual price differs from `WAD`, the source starts with an inconsistent TVL that converges only through the EMA, potentially misweighting or excluding it. The seed is irrelevant for NG sources because `_ema_tvl()` reads `D_oracle()` directly.

> **Code Corrected**

> In **Version 2**, legacy sources start with zero stored TVL instead of an LP-supply seed. Subsequent EMA observations use `_legacy_tvl()`, which accounts for the virtual price through `totalSupply() * get_virtual_price() // WAD`.

#012 Only the First Two Coins of an NG Pool Can Be Used

Informational **Version 1** **Specification Changed**

In AggregateStablePrice4, `add_price_pair()` only checks `coins(0)` and `coins(1)`, while `_get_p()` always reads `price_oracle(0)`, which represents coin 1 relative to coin 0. Consequently, coins at indices 2 and above in a StableSwap-NG pool cannot be used by the aggregator.

> **Specification Changed**
>
> In **Version 2**, the NatSpec of `add_price_pair()` explicitly limits the intended sources to two-coin StableSwap pools pairing the stablecoin with a redeemable asset.

#013 Price-Pair Metadata Is Loaded Twice per Active Source

Informational **Version 1** **Acknowledged**

In AggregateStablePrice4, `_ema_tvl()` loads all three storage slots of `PricePair` for every source but does not use `is_inverse`. `_active_sources()` later loads all three slots again for each active source before calling `_get_p()`.

> **Acknowledged**
>
> Curve acknowledges the finding.

#014 Removed Sources Remain Visible Through Public Getters

Informational **Version 1** **Code Corrected**

`remove_price_pair()` decreases the private `n_price_pairs` counter but does not clear the former last entries in the public `price_pairs` and `last_tvl` arrays. Their getters can therefore return data for sources that are no longer used by the aggregator.

As there is no public getter for `n_price_pairs`, integrators reading these arrays cannot directly identify their active prefix. The count can nevertheless be inferred from the length returned by `ema_tvl()` or reconstructed from events.

> **Code Corrected**
>
> In **Version 2**, `n_price_pairs` is public, and `remove_price_pair()` clears the vacated tail entries in both `price_pairs` and `last_tvl`.

#015 Reseeding an Empty NG Pool Reinitializes Its D Oracle

Informational **Version 1** **Acknowledged**

The first deposit into a zero-supply StableSwap-NG pool initializes its averaged `D`. Thus, a party able to empty and reseed a pool can set `D_oracle()` to the new deposit's `D`. If the deposit is withdrawn in the same block, the oracle retains this value and decays afterward, while AggregateStablePrice4 uses it as the source's TVL.

Acknowledged

Curve states:

> StableSwap-NG behaviour outside the aggregator: the first deposit into an empty pool writes both the last and the averaged D directly (`last_D_packed = pack_2(D1, D1)`), and a same-block withdrawal leaves the averaged D in place, so `D_oracle()` starts at the deposit's D and decays with `D_ma_time`. A pool that can be fully emptied is already below `min_liquidity` and inactive, the attacker has to deposit at least `min_liquidity` to activate it, and in the meantime the source's influence is bounded by the cap and the exponential penalty exactly as in findings 004 and 005.

#016 Shared Constants Are Duplicated Across Modules

Informational **Version 1** **Acknowledged**

AggregateStablePrice4, `capped_share`, and `weighted_price` each define `WAD` and `MAX_N` independently. Changes must therefore be repeated in all three places. In particular, different `WAD` values would compile while making the modules use different scales.

Acknowledged

Curve intentionally keeps `capped_share` and `weighted_price` self-contained for reuse by other oracles and accepts the duplicated constants.

#017 The Same Pool Can Be Configured as Several Independent Price Sources

Informational **Version 1** **Code Corrected**

AggregateStablePrice4 allows the admin to register the same pool more than once. Each active copy contributes another observation of the pool's price and another TVL entry, increases the active-source count, and receives its own per-source cap. A duplicated pool can therefore receive more than the intended cap in total and has additional weight in the reference-price calculation.

Code Corrected

In **Version 2**, `add_price_pair()` checks all registered sources and rejects a pool whose address is already present. Removal keeps the registered sources contiguous and clears the vacated slot, allowing a removed pool to be registered again.

#018 A Very Low Liquidity Floor Can Block Price Calculations

Informational **Version 2** **Code Corrected**

In `AggregateStablePrice4`, the admin can set `min_liquidity` to any positive value. An extremely low floor can admit a source whose share of total TVL is too small to produce a nonzero weight at 18-decimal precision. Nevertheless, that source participates in selecting the minimum exponential penalty. If it is closest to the reference price, it receives a multiplier of one but retains zero weight. All positive-weight sources can then receive penalties that round their weights to zero, leaving the final weighted average to divide by zero.

For example, set `min_liquidity = 1` and `sigma = 1e16`, with five sources holding $1M each priced at 0.93, 0.93, 0.93, 1.105, and 1.105, plus a 1-wei-TVL source priced at 1.00. The large sources each receive approximately 20% base weight and produce a reference price of 1.00. The tiny source has zero weight but sets the penalty baseline, while the large sources' exponential multipliers round to zero. Both `price()` and a subsequent `price_w()` checkpoint revert, blocking oracle consumers.

For realistic pool TVLs, this requires an extremely low floor. The default $100k floor excludes the tiny source needed for this example.

Code Corrected

In **Version 3**, `exp_penalized_price()` ignores zero-weight sources when computing the penalty baseline.

#019 Immutable `SIGMA` Accepts a Wide, Partly Redundant Range

Informational **Version 1** **Code Corrected**

In `weighted_price`, the immutable `SIGMA` may be set anywhere from `1e9` to `1e18`. The penalty exponent divides each source's squared price deviation by `SIGMA**2 // WAD`, so the selected value directly changes the penalty. Near the lower bound, many different settings have exactly the same effect: every value from `1e9` through `1_414_213_562` produces a denominator of one. The code does not document an intended value, and `SIGMA` cannot be changed after deployment, so a poorly chosen value can only be corrected by redeploying. The intended value for crvUSD could be documented, and the constructor could enforce a narrower range.

Code Corrected

In **Version 2**, `sigma` is stored and can be changed by the admin through `set_sigma()` without redeployment or state migration. The constructor and setter validate the range `[1e9, 1e18]` and emit `SetSigma`. Curve retains this range as an arithmetic bound and considers the coarse denominator near its lower end irrelevant for realistic settings.

#020 Pending Removals Can Delete the Wrong Price Source

Informational **Version 2** **Code Corrected**

In `AggregateStablePrice4`, `remove_price_pair()` identifies a source only by index. Admin removals typically follow a slow governance vote with fixed calldata, while the emergency admin can act quickly through a multisig. A removal can therefore change the target of another pending removal, causing it to delete an unintended source or revert if the index is no longer valid. Such a revert also blocks other actions bundled in the same governance transaction.

Code Corrected

In **Version 3**, `remove_price_pair()` identifies the source by its pool address instead of its index.

#021 Source Caps Can Give Smaller Pools a Combined Majority

Informational **Version 1** **Acknowledged**

AggregateStablePrice4 caps each source's weight to limit reliance on any one pool. Weight above the cap is distributed among the other active sources according to their TVLs.

With source TVLs of $10 billion, $100,000, and $100,000, the default 45% cap produces weights of 45%, 27.5%, and 27.5%. The two smaller sources therefore have a combined majority despite holding approximately 0.002% of the total liquidity.

These weights also determine the reference price used by the exponential penalty. If the deeper source reports 1.00 and both smaller sources report 1.10, with `sigma = 1e16` the aggregator reports approximately 1.099996.

Similarly, all capped sources receive the same weight regardless of their TVL, so when two capped pools report prices several `sigma` apart, for example with weights of 45%, 45%, and 10%, the single small source decides which price the aggregator follows.

While capping source weights promotes diversity and protects against black swan events such as a depeg of one large pool, it can also give disproportionate influence to very small sources, weakening the manipulation resistance provided by deeper pools.

Acknowledged

Curve accepts this trade-off to prevent a single deep pool from dominating the price. Governance selects reputable, liquid sources, and the emergency admin can remove sources whose liquidity has degraded. Curve considered and rejected uncapped reference weights because they restore reliance on a dominant pool.

#022 The Liquidity Floor and TVL Averaging Window Cannot Be Adjusted

Informational **Version 1** **Code Corrected**

AggregateStablePrice4 fixes `MIN_LIQUIDITY` at `100_000e18` and `TVL_MA_TIME` at 50,000 seconds, while sigma is selected at deployment and the share cap is configurable. Neither value can be changed after deployment.

Code Corrected

In **Version 2**, the admin can change `min_liquidity` and `tvl_ma_time` through `set_min_liquidity()` and `set_tvl_ma_time()`. Both setters reject zero and emit an event. Deployment defaults remain `100_000e18` and 50,000 seconds, respectively.

#023 TVL Observations Have No Relative Change Bound

Informational **Version 1** **Acknowledged**

AggregateStablePrice4 accepts legacy TVL observations from `totalSupply() * get_virtual_price() // WAD` and NG observations from `D_oracle()` without an absolute or per-update bound. A caller can repeatedly manipulate a legacy observation around `price_w()` calls and gradually move its EMA, while an NG observation is used directly.

> **Acknowledged**
>
> Curve intentionally leaves TVL changes unbounded so genuine liquidity outflows can promptly reduce a source's weight. It considers the existing EMA sufficient for inflows and notes that a per-update bound can be bypassed through repeated checkpoints. NG pools provide their own averaged `D_oracle()` observations.

#024 TVL Window Changes Can Abruptly Reweight or Freeze Legacy Sources

Informational **Version 2** **Code Corrected**

In `AggregateStablePrice4`, the admin can change the TVL averaging window for legacy pools through `set_tvl_ma_time()`.

The new window applies retroactively to all time since the last checkpoint. Shortening it can therefore immediately increase a source's weight without any change in pool liquidity. For a source starting from zero, switching to a one-second window after 10% of the original window has elapsed can raise its computed TVL from about 9.5% to almost 100% of the pool's current TVL.

The window also has no upper bound. Setting `tvl_ma_time` to `10**30` makes daily `price_w()` calls round the EMA adjustment to zero, freezing stored legacy TVLs. Sources starting from zero then remain inactive, and `price()` returns the `WAD` placeholder if no other source qualifies.

> **Code Corrected**
>
> In **Version 3**, `set_tvl_ma_time()` checkpoints the sources before applying the new window, and `tvl_ma_time` is bounded by 30 days.

8 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.