ClarityAlliance

BITFLOW STABLESWAP MIDPOINT SECURITY REVIEW

Conducted by:
KRISTIAN APOSTOLOV, ALIN BARBATEI (ABA), MARCHEV

MARCH 13TH, 2025

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Midpoint

1. About Clarity Alliance

Clarity Alliance is a team of expert whitehat hackers specialising in securing protocols on Stacks.

They have disclosed vulnerabilities that have saved millions in live TVL and conducted thorough reviews for some of the largest projects across the Stacks ecosystem.

Learn more about Clarity Alliance at clarityalliance.org.

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Midpoint

2. Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Clarity Alliance to perform a security assessment.

This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Clarity Alliance’s position is that each company and individual are responsible for their own due diligence and continuous security. Clarity Alliance’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree
to analyze.

The assessment services provided by Clarity Alliance are subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis.

Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third parties. Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Clarity Alliance does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Midpoint

3. Introduction

A time-boxed security review of Bitflow Stableswap, where Clarity Alliance reviewed the scope and provided insights on improving the protocol.

4. About Bitflow StableSwap

Bitflow StableSwap is the first protocol designed to enable users to efficiently swap stable assets, including stablecoins, within the Bitcoin ecosystem. It operates on the Stacks layer, a platform specifically designed to facilitate smart contracts and decentralized applications on Bitcoin.

CONTENTS

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1 Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2 Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

CONTENTS

6. Security Assessment Summary

Scope

The following contracts were in the scope of the security review:

- `contracts/stableswap-pool-trait-v-1-1.clar`
- `contracts/stableswap-emissions-stx-ststx-stx-v-1-1.clar`
- `contracts/stableswap-core-v-1-1.clar`
- `contracts/stableswap-staking-stx-ststx-v-1-1.clar`
- `contracts/stableswap-swap-helper-v-1-1.clar`
- `contracts/token-stx-v-1-1.clar`
- `contracts/sip-010-trait-ft-standard-v1-1-1.clar`
- `contracts/stableswap-pool-stx-ststx-v1-1-1.clar`

Initial Commit Reviewed:
2d61eec056f1e0b1fcd5ea458c84aa077ace3410

Intermediate Commit Reviewed:
459ddf7f921f392695d790cbbf05380a30f730d0

Final Commit After Fixes:
66400b5a1e6c6246cfd88f6521852f1e24c9aa26

CONTENTS

7. Executive Summary

Over the course of the security review, Kristian Apostolov, Alin Barbatei (ABA), Marchev engaged with - to review Bitflow StableSwap. In this period of time a total of **12** issues were uncovered.

Protocol Summary

Protocol Name	Bitflow StableSwap
Date	March 13th, 2025

Findings Count

Severity	Amount
Critical	1
High	3
Medium	2
Low	2
QA	4
Total Findings	**12**

CONTENTS

Summary of Findings

ID	Title	Severity	Status
[C-01]	Malicious Public Pool Creators Can Steal All Y Tokens in Public Pools	Critical	Resolved
[H-01]	Depreciating Midpoint Adjusted Pairs Will Block User Funds	High	Resolved
[H-02]	Midpoint Adjustment Vulnerability Allows Token Extraction	High	Resolved
[H-03]	Liquidity Providers Can Drain Pools by Exploiting the Liquidity Mechanism in Pairs with Elevated X Token	High	Resolved
[M-01]	Swap Amount Is Incorrectly Inversely Adjusted Via Midpoint	Medium	Resolved
[M-02]	Midpoint Variable and Factor Must Be Changed Simultaneously	Medium	Resolved
[L-01]	Midpoint Constraint Only Allows Unidirectional Price Adjustments	Low	Resolved
[L-02]	Midpoint Manager Unable to Manage Midpoint Reversed Flag	Low	Resolved
[QA-01]	Fixed Liquidity Provision Ratio Despite Dynamic Exchange Rates	QA	Acknowledged
[QA-02]	Misleading Error for Midpoint Factor Validation	QA	Resolved
[QA-03]	Midpoint Factor Not Logged During Swaps	QA	Resolved
[QA-04]	Simplification of Midpoint Bilateral Price Adjustment Mechanism	QA	Resolved

CONTENTS

8. Findings

8.1. Critical Findings

[C-01] Malicious Public Pool Creators Can Steal All Y Tokens in Public Pools

Description

The protocol permits pool creation by both protocol administrators and any user if `public-pool-creation` is enabled. While protocol administrators and the midpoint managers they appoint are considered trustworthy, this assumption does not extend to untrusted public users.

When `public-pool-creation` is enabled, a malicious user can create a pool and appoint an arbitrary midpoint manager. This midpoint manager can exploit the midpoint configuration to artificially inflate the value of token `X` relative to token `Y`, allowing them to steal all `Y` tokens in the pool through a `swap-x-for-y` transaction.

Specifically, the protocol enforces the invariant `midpoint >= midpoint-factor`. This invariant allows the midpoint manager to set an excessively large `midpoint` and a very small `midpoint-factor`, resulting in an artificially reduced `X` token balance and inflated `X` token value. The critical mechanism is the formula used to scale the `X` token balance:

```
(x-balance-midpoint-scaled (/
  (* x-balance-scaled midpoint-factor) midpoint)) ;; @audit Could be vastly reduced in
```

By manipulating the midpoint values, a malicious midpoint manager can make the `X` balance appear extremely small, enabling them to execute a `swap-x-for-y` and steal virtually all `Y` tokens from the pool in exchange for a negligible amount of `X`.

Example Scenario

1. Alice, a malicious actor, creates a public pool with `USDC` and `stUSDC` as tokens and appoints herself as the midpoint manager.
2. Users provide liquidity to the pool:

```
USDC = 1000
stUSDC = 1000
```

3. Alice configures the midpoint values as follows:

```
midpoint = u99999999999
midpoint-factor = u1
```

4. Alice swaps 1 USDC for stUSDC via `swap-x-for-y`:

```
x-balance-midpoint-scaled = (1000 * 1) / 99999999999 ≈ 1.0 × 10^-8
```

This manipulation makes USDC appear vastly overvalued. Alice receives approximately 999.106618 stUSDC in exchange for 1 USDC, leaving liquidity providers at a loss.

CONTENTS

ClarityAlliance
Security Review

Bitflow Stableswap Midpoint

Recommendation

The mitigation approach depends on the protocol's business requirements. One possible solution is to restrict public pool creators from specifying a midpoint manager and instead configure the protocol admin as the default midpoint manager for public pools.

If public pools must remain permissionless, consider the following safeguards:

1. Restrict the `midpoint / midpoint-factor` ratio to a maximum acceptable value, configurable by the protocol admin.
2. Introduce a cooldown period after midpoint configuration changes, allowing liquidity providers to withdraw their funds in case of malicious settings.

These measures can reduce the risk while maintaining flexibility for the protocol's use cases.

8.2. High Findings

[H-01] Depreciating Midpoint Adjusted Pairs Will Block User Funds

Description

The midpoint mechanism allows the midpoint manager (or an admin) to adjust the default 1:1 exchange ratio for a Stableswap Core pool. By increasing the midpoint, users should receive more of token Y when swapping token X for it. Conversely, users will receive less of token X when swapping token Y for it.

This mechanism is beneficial for liquid staking tokens, as their exchange ratio should increase over time as they generate yield. For example, a midpoint manager contract can be created to capture StackingDAO's `STX:stSTX` exchange ratio and use it as the midpoint for an `STX-stSTX` pool.

The `stableswap-core-v-1-1` implementation is designed to support scenarios where the X token has a higher ratio than the Y token and vice versa. This means it should accommodate both `STX-stSTX` and `stSTX-STX` adjusted pools. While swaps support both pairing directions, adding and withdrawing liquidity results in reversion.

Due to the incorrect implementation of adding and withdrawing liquidity, users may find their LPs blocked. Consider the following scenario:

- The `XYZ:stXYZ` pool is newly deployed with an initial 1:1 ratio, expected to fluctuate.
- Users add liquidity when the midpoint is set neutrally at 1:1.
- Market conditions naturally increase the `stXYZ` value relative to `XYZ`, causing the midpoint logic to decrease the value of `XYZ` as currently implemented.

Users are then unable to withdraw their LPs (or add more) due to an underflow in the `add-liquidity` and `withdraw-liquidity` functions, effectively blocking their LPs until market conditions return to a 1:1 ratio or the team manually sets the midpoint ratio to 1:1, resulting in a loss for the users.

The issue arises in `add-liquidity` due to the calculation of the `midpoint-discount-value`:

```
;; Calculate midpoint discount amount
(midpoint-value-a (if midpoint-reversed midpoint-factor midpoint))
(midpoint-value-b (if midpoint-reversed midpoint midpoint-factor))
(midpoint-discount-value (- midpoint-value-b (/
  (* midpoint-value-b midpoint-value-b) midpoint-value-a)))
```

CONTENTS

The subtraction `(- midpoint-value-b (/ (* midpoint-value-b midpoint-value-b) midpoint-value-a))` becomes negative if `midpoint-value-b` is ever greater than `midpoint-value-a`, causing a reversion.

In the `withdraw-liquidity` function, an identical calculation is performed for the adjusted value, with the variable named `midpoint-addition-value`.

Recommendation

The midpoint adjusting logic in both the `add-liquidity` and functions `withdraw-liquidity` needs to be modified to correctly support inversely valued tokens.

CONTENTS

[H-02] Midpoint Adjustment Vulnerability Allows Token Extraction

Description

The inclusion of the midpoint variable can expose pools with tokens of varying values to an attack. An attacker can exploit the imbalanced liquidity provision mechanism to extract value from the pool, even with a liquidity fee in place. This is particularly concerning for pools where one token naturally appreciates against the other, such as STX/stSTX.

To address this, the current Stableswap implementation ensures that both withdrawing and adding liquidity are correctly scaled by the midpoint.

However, value extraction remains possible in scenarios of high volatility and significant price ratio increases.

Consider the following scenario:

- 1 stSTX = 1.2 STX (configured via the midpoint)
- Due to an unexpected event, the `stSTX` token value increases to 1.3 STX in real valuation
- The midpoint manager initiates a transaction to call the set-midpoint function with the new `1.3` ratio
- An attacker observes the `set-midpoint` transaction and sandwiches it with a single-sided STX `add-liquidity` call and a `withdraw-liquidity`
- action to extract more `STX` equivalent in `stSTX` than was added, due to the `0.1` difference in ratio.

The attack is profitable when the gains from the `stSTX` conversion rate increase offset the liquidation fee.

Note: The midpoint effectively acts as an oracle update and inherits some of its issues. The described attack operates on the same principle as self-liquidations in a borrowing and lending protocol (5.1.2).

Similar to other oracle-related issues, the current implementation lacks a staleness check for midpoint values, which could lead to the use of outdated midpoint values.

Recommendation

While oracle-equivalent frontrunning attacks cannot be fully mitigated, the risk of this attack can be reduced by diminishing the gains from midpoint adjustments in `withdraw-liquidity`. This value should also be adjustable by the midpoint manager. For simplicity, it can be implemented as a separate midpoint value for withdrawals.

CONTENTS

[H-03] Liquidity Providers Can Drain Pools by Exploiting the Liquidity Mechanism in Pairs with Elevated X Token

Description

The introduction of the variable midpoint can render pools with tokens of differing values susceptible to an attack. In such scenarios, an attacker can exploit the imbalanced liquidity provision mechanism to extract value from the pool, even in the presence of a liquidity fee. This issue is particularly concerning for pools where one token naturally appreciates against the other, such as STX/stSTX.

To address this, the current Stableswap implementation ensures that both withdrawing and adding liquidity are correctly scaled by the midpoint.

However, the existing implementation only effectively compensates for pools where `Y` is the more valuable token, such as in `STX-stSTX`. For pools where `X` is the yield-bearing and more powerful token, such as `stSTX-STX`, the pool remains vulnerable.

Recommendation

Implement a correct midpoint algorithm adjustment or discontinue support for `X->Y` pools where `X` is more valuable than `Y`.

8.3. Medium Findings

[M-01] Swap Amount Is Incorrectly Inversely Adjusted Via Midpoint

Description

When a swap is initiated, either `X -> Y` or `Y -> X`, the `stableswap-core-v-1-1` implementation adjusts the perceived value of a token through midpoint price amplification. This is achieved by artificially increasing the balance of the target token, causing the Stableswap algorithm to perceive an imbalance. As a result, more or fewer paired tokens are returned, effectively simulating a price increase.

During a swap, the midpoint amplification is applied to both the existing pool balance and the newly swapped amount of the amplified token:

```
;; Swap x token for y token via a pool
    (midpoint-value-a (if midpoint-reversed midpoint-factor midpoint))
    (midpoint-value-b (if midpoint-reversed midpoint midpoint-factor))
    (dx-midpoint-scaled (/ (* dx-scaled midpoint-value-b) midpoint-value-a))
    (x-balance-midpoint-scaled (/
      (* x-balance-scaled midpoint-value-a) midpoint-value-b))

    ;; Swap y token for x token via a pool
    (midpoint-value-a (if midpoint-reversed midpoint midpoint-factor))
    (midpoint-value-b (if midpoint-reversed midpoint-factor midpoint))
    (dy-midpoint-scaled (/ (* dy-scaled midpoint-value-b) midpoint-value-a))
    (y-balance-midpoint-scaled (/
      (* y-balance-scaled midpoint-value-a) midpoint-value-b))
```

However, in both swap directions, the balance to be swapped is incorrectly scaled in the opposite direction of the intended design.

Example:

- For a pool with `midpoint-reversed=false`, `midpoint=1_200_000`, and `midpoint-factor=1_000_000`
- Swapping 1000 `stSTX` swap amount (`dx-midpoint-scaled`) being considered as `1000 * 1_000_000 / 1_200_000 = 833`, which is an incorrect reduction in perceived value.
- The existing pool balance of 10,000 `stSTX` (`x-balance-midpoint-scaled`) will be correctly considered as `10_000 * 1_200_000 / 1_000_000 = 12000`, equating to 12000 tokens.

Depending on the swap amount relative to the existing balance, the incorrect amplification could have a negligible or severe impact on the resulting paired and swapped amount.

CONTENTS

Recommendation

In `get-dx` and `swap-y-for-x`, change:

```
(dy-midpoint-scaled (/ (* dy-scaled midpoint-value-b) midpoint-value-a))
```

to:

```
(dy-midpoint-scaled (/ (* dy-scaled midpoint-value-a) midpoint-value-b))
```

In `get-dy` and `swap-x-for-y`, change:

```
(dx-midpoint-scaled (/ (* dx-scaled midpoint-value-b) midpoint-value-a))
```

to:

```
(dx-midpoint-scaled (/ (* dx-scaled midpoint-value-a) midpoint-value-b))
```

[M-02] Midpoint Variable and Factor Must Be Changed Simultaneously

Description

In the midpoint logic, two variables are crucial when calculating the increase:

- `midpoint` : Initially intended as the numerator of the final percent increase.
- `midpoint-factor` : The denominator of the final midpoint percent increase, equivalent to a variable total BPS.

Excluding reverse operations, which use the `midpoint-reversed` flag, both the `midpoint` and `midpoint-factor` variables need to be changed simultaneously when transitioning from one granularity to another.

Example:

- `midpoint=120` , `midpoint-factor=100` , equivalent to a 1.2 increase.
- If the midpoint manager wants to set it to 1.215, both variables need to be updated as follows:
  - `midpoint=1215` , `midpoint-factor=1000`

Each of the `midpoint` and `midpoint-factor` variables has a separate setter, requiring the manager to call them sequentially, in a non-atomic manner. If a swap occurs between these setter calls, it will result in a highly distorted value and may even lead to funds being extracted from the pool by overinflating one token relative to another.

Recommendation

Implement a single setter for both the `midpoint` and `midpoint-factor` variables.

8.4. Low Findings

[L-01] Midpoint Constraint Only Allows Unidirectional Price Adjustments

Description

The current implementation enforces an invariant where the `midpoint` must be greater than or equal to the `midpoint_factor`

```
;; Assert that midpoint is greater than or equal to midpoint-factor
(asserts! (>= midpoint midpoint-factor) ERR_INVALID_MIDPOINT)
```

This constraint means that the midpoint mechanism can only adjust prices in one direction, specifically making token `X` more valuable relative to token `Y`. However, for pools involving staked tokens (e.g., STX-stSTX), the typical expectation is for stSTX to increase in value over time compared to STX as it accrues staking rewards.

The current constraint prevents this desired behavior, forcing protocols to create separate pools in the reverse order (e.g., stSTX-STX) to achieve the intended price adjustment direction. This results in unnecessary liquidity fragmentation across multiple pools for the same token pair.

Recommendation

Consider one of the following approaches:

1. Allow the midpoint to be configured in both directions by removing the `>=` constraint between `midpoint` and `midpoint_factor`. This change would provide maximum flexibility for pools to adjust prices in either direction.
2. If pools are expected to always be created in the order of Token → Staked Token, reverse the constraint to `midpoint-factor >= midpoint`. This adjustment would allow staked tokens to appreciate relative to their unstaked versions.

The first option is recommended as it offers the most flexibility for different use cases. However, if there are specific security considerations around the midpoint mechanism that necessitate maintaining a one-way constraint, then option 2 would better align with the common staked token use case.

CONTENTS

[L-02] Midpoint Manager Unable to Manage Midpoint Reversed Flag

Description

In the `stableswap-core-v-1-1` contract, the `set-midpoint-reversed` function is responsible for setting the reversed midpoint flag.

Currently, only an admin has the authority to set this value. This is insufficient, as the midpoint manager should also have the ability to modify this value, given that they can adjust other midpoint-related settings.

Recommendation

Modify the caller verification in the `stableswap-core-v-1-1::set-midpoint-reversed` function to include the `midpoint-manager` as an authorized caller:

```
(asserts! (or (is-some (index-of (var-get admins) caller))
(is-eq midpoint-manager caller)) ERR_NOT_AUTHORIZED)
```

CONTENTS

8.5. QA Findings

[QA-01] Fixed Liquidity Provision Ratio Despite Dynamic Exchange Rates

Description

The `add-liquidity` function in the stableswap implementation currently mandates that liquidity be added in an approximately 1:1 ratio between tokens. This requirement does not consider the `midpoint` and `midpoint_factor` when calculating the ideal pool balances.

This becomes problematic when one token appreciates relative to the other (such as stSTX appreciating against STX due to staking rewards). For example:

1. Initially, 1 STX equals 1 stSTX, so liquidity providers (LPs) supply liquidity in a 1:1 ratio (e.g., 10,000 STX and 10,000 stSTX).
2. After a midpoint adjustment makes stSTX twice as valuable (1 STX = 0.5stSTX), the function still requires a 1:1 liquidity provision.
3. Consequently, LPs must provide twice the stSTX value necessary based on the actual exchange rate.

The current implementation forces liquidity providers to over-commit the more valuable token, leading to capital inefficiency as excess tokens are locked in the pool beyond what's needed to facilitate swaps at the intended exchange rate.

Recommendation

This behavior is an inherent characteristic of the chosen midpoint design. Since the protocol is designed to maintain stable prices through its core stableswap algorithm, the 1:1 liquidity provision ratio is a fundamental part of its operation. No mitigation is recommended as the current implementation is functioning as intended. However, users and integrators should be aware of this characteristic when deciding to provide liquidity to these pools, especially in scenarios where one token is expected to appreciate significantly relative to the other.

CONTENTS

[QA-02] Misleading Error for Midpoint Factor Validation

Description

The `set-midpoint-factor` function checks that the midpoint value is greater than or equal to the factor being set:

```
;; Assert that midpoint is greater than or equal to factor
(asserts! (>= midpoint factor) ERR_INVALID_MIDPOINT)
```

However, the error `ERR_INVALID_MIDPOINT` is misleading in this context. The validation pertains to the relationship between the midpoint and the factor, yet the error name implies an issue solely with the midpoint. This can cause confusion, as the function's primary focus is on updating and validating the factor.

Recommendation

Revise the error name to more clearly reflect the relationship being validated, such as `ERR_MIDPOINT_FACTOR_EXCEEDS_MIDPOINT`.

CONTENTS

[QA-03] Midpoint Factor Not Logged During Swaps

Description

During swap operations, the implementation logs the `midpoint` value but fails to include the `midpoint-factor` , which is crucial for calculating the midpoint-scaled balances. Consequently, the logs offer an incomplete view of the calculation process. Since the midpoint can have a fractional component due to its reliance on the divisor (midpoint factor), omitting this value makes it challenging to debug or monitor the protocol effectively.

Recommendation

Modify the implementation to log the `midpoint-factor` along with the `midpoint` during swap events. This will provide a complete context for the calculation, enhancing observability and facilitating debugging.

CONTENTS

[QA-04] Simplification of Midpoint Bilateral Price Adjustment Mechanism

Description

The current configuration for modifying the midpoint involves three variables, each with its own setter:

- `midpoint` : Initially intended as the numerator for the final percentage increase.
- `midpoint-factor` : Serves as the denominator for the final midpoint percentage increase, equivalent to a variable total BPS.
- `midpoint-reversed` : Determines whether to consider the `midpoint` and `midpoint-factor` values in reverse. In a `X -> Y` pool, this would mean Y is the inflated token, not `X`.

The existing midpoint setup includes redundant logic to allow for a midpoint reversal. Both `midpoint` and `midpoint-factor` can be directly set with mirrored values, achieving the same effect without needing a `midpoint-reversed` variable.

The only additional requirement is to have a single function that sets both `midpoint` and `midpoint-factor` values simultaneously, preventing transactions from occurring during the transition of the midpoint value (as noted in a separate finding).

Recommendation

Eliminate the `midpoint-reversed` logic. Rename `midpoint` to `midpoint numerator` and `midpoint-factor` to `midpoint-denominator`. In the code, replace the `midpoint-value-a` placeholder with the `midpoint-numerator` value, and replace `midpoint-value-b` with the `midpoint-` variable.

Create a single setter function for `midpoint-numerator` and `midpoint-denominator` that sets both values simultaneously, ensuring they are both greater than 0.

This approach significantly simplifies the code. To reverse the midpoint price increase, instead of using `midpoint-numerator=1_200_000` and `midpoint-denominator=1_000_000`, simply reverse them: `midpoint-numerator=1_000_000` and `midpoint-denominator=1_200_000`. This achieves the same effect without using an intermediary `midpoint-reversed` value or placeholder variables.