Quantstamp Security Assessment Certificate

December 10th 2020 — Quantstamp Verified

Saddle Finance

This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

Type	StableSwap implementation
Auditors	Sebastian Banescu, Senior Research Engineer Kevin Feng, Blockchain Researcher
Timeline	2020-10-28 through 2020-12-09
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	StableSwap whitepaper
Documentation Quality	Low
Test Quality	High

Source Code

Repository	Commit
saddle-contract	83491b3

Total Issues	19 (12 Resolved)
High Risk Issues	1 (1 Resolved)
Medium Risk Issues	4 (2 Resolved)
Low Risk Issues	5 (4 Resolved)
Informational Risk Issues	8 (5 Resolved)
Undetermined Risk Issues	1 (0 Resolved)

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Quantstamp has performed a security review of the Saddle Finance implementation of StableSwap. It is important to note that this implementation is ported from `SwapTemplateBase.vy` in the Curve Finance contracts, which was used as a reference during the review. In total 14 security issues spanning across all severity levels were identified, along with a few deviations from the specification, code documentation issues and best practice issues. Due to the poor documentation we were not able to determine how the developers have derived some of the implemented formulas from the StableSwap whitepaper. Additionally, we have noticed that all tests in the current test suite use exactly 2 tokens in the pool. We strongly recommend adding more tests that use 3 or more tokens and addressing all identified issues before deploying the code in production.

Update: Quantstamp has reviewed the changes to the code corresponding to commit hash `5a56e24` and has updated the status of all 14 issues which were previously identified. Additionally, we have identified 4 new issues in the newly added code. These new issues were added after the existing issues and their identifiers are between QSP-15 to QSP-18.

Update: Quantstamp has reviewed the changes to the code corresponding to commit hashes `ebec9fd`, `759c028`, `33baaaa`. The main focus of these iterations was improving the existing test suite to verify the impact of QSP-15 and the newly added QSP-19.

ID	Description	Severity	Status
QSP-1	Incorrect computation in `getD`	High	Fixed
QSP-2	Integer Overflow / Underflow	Medium	Fixed
QSP-3	Missing input validation	Medium	Fixed
QSP-4	Virtual price calculation is not future-proof	Low	Fixed
QSP-5	Increased loss of precision due to multiplication after division	Low	Acknowledged
QSP-6	Crude check of contract address does not work	Low	Fixed
QSP-7	Checks-effects-interactions pattern violated in `addLiquidity`	Low	Fixed
QSP-8	Privileged Roles and Ownership	Informational	Acknowledged
QSP-9	Unlocked Pragma	Informational	Fixed
QSP-10	Methods with different names and same implementation	Informational	Fixed
QSP-11	Missing assertion in `removeLiquidityImbalance`	Informational	Fixed
QSP-12	Block Timestamp Manipulation	Informational	Acknowledged
QSP-13	Accidental overwriting of multipliers	Informational	Acknowledged
QSP-14	Allowed amount could be higher than pool cap	Undetermined	Acknowledged
QSP-15	The value of the A parameter can be influenced by `block.timestamp`	Medium	Acknowledged
QSP-16	`stopRampA` may be called repeatedly even after a ramp has ended	Low	Fixed
QSP-17	Ramping can be started while previous ramp is still ongoing	Informational	Fixed
QSP-18	Integer Overflow	Informational	Fixed
QSP-19	Loss-Making Updates to A	Medium	Acknowledged

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.6.13

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Incorrect computation in `getD`

Severity: *High Risk*

Status: Fixed

File(s) affected: `SwapUtils.sol`

Description: The `SwapUtils.getD` function contains an incorrect `add(1)` for the denominator of `dP` inside the inner `for`-loop, on L241. This will lead to an incorrect value of `D` being returned every time `SwapUtils.getD` is called.

Recommendation: Remove the `add(1)` mentioned above.

QSP-2 Integer Overflow / Underflow

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `SwapUtils.sol`

Description: Integer overflow/underflow occur when an integer hits its bit-size limit. Every integer has a set range; when that range is passed, the value loops back around. A clock is a good analogy: at 11:59, the minute hand goes to 0, not 60, because 59 is the largest possible minute. Integer overflow and underflow may cause many unexpected kinds of behavior and was the core reason for the `batchOverflow` attack.
The subtraction inside `SwapUtils.withdrawAdminFees` does not use `SafeMath`, which could cause an underflow if `token.balanceOf(address(this)) < self.balances[i]`. The result of the underflow would then be assigned to the `uint256 balance` variable, which would be positive and therefore always pass the check on the subsequent line: `if (balance > 0)`. A similar underflow issue occurs on L504 inside `SwapUtils.getAdminBalance`.

Recommendation: Use `SafeMath.sub` instead of primitive arithmetic subtraction.

QSP-3 Missing input validation

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `Swap.sol`, `Allowlist.sol`

Description: The following functions are missing checks for user input:

1. **[Fixed]** The `Swap.getToken` function does not check if the value of the `index` parameter is lower than the length of the `pooledTokens` array.
2. **[Fixed]** The `Swap.getTokenBalance` function does not check if the value of the `index` parameter is lower than the length of the `pooledTokens` array.
3. **[Fixed]** The `Swap.calculateSwap` function does not check if value of the 2 `tokenIndex{From | To}` parameters are lower than the length of the `pooledTokens` array. It also does not check if the value of the `dx` input parameter is lower than the amount of tokens that the user actually has.
4. **[Fixed]** The `Swap.calculateRemoveLiquidity` function does not check if value of the `amount` parameter is larger than the total supply of the `swapStorage.lpToken`.
5. **[Fixed]** The `Swap.calculateRemoveLiquidityOneToken` function does not check if the value of the `tokenIndex` parameter is lower than the length of the `pooledTokens` array. It also doesn't check if the value of the `tokenAmount` is lower than the actual amount available for that token.
6. **[Fixed]** The `Swap.getAdminBalance` function does not check if the value of the `index` parameter is lower than the length of the `pooledTokens` array.
7. **[Fixed]** The `Swap.swap` function does not check if value of the 2 `tokenIndex{From | To}` parameters are lower than the length of the `pooledTokens` array. It also does not check if the value of the `dx` input parameter is lower than the amount of tokens that the user actually has.
8. **[Fixed]** There should be checks for `poolAddress != address(0x0)` in the functions `Allowlist.setPoolAccountLimit`, and `Allowlist.setPoolCap`.
9. **[Fixed]** The `Swap.constructor` does not check if the length of the `_pooledTokens.length > 1`. This is an implicit assumption, which should be made explicit.
10. **[Fixed]** The `Swap.constructor` does not check if all the addresses provided in `_pooledTokens` are different. This would allow pools with multiple tokens, where the tokens have the same address.

The consequences of an exploit of the aforementioned items vary. However, some of the items mentioned above could significantly impact the reputation of the project and should be therefore addressed.

Recommendation: Add `require` statements in the functions enumerated above, which should check that the input arguments are within bounds and indicate appropriate error messages otherwise.

QSP-4 Virtual price calculation is not future-proof

Severity: *Low Risk*

Status: Fixed

File(s) affected: `SwapUtils.sol`

Description: The `SwapUtils.getVirtualPrice` claims to calculate "the virtual price, scaled to the POOL_PRECISION". In order to do this it multiplies `D` by `10 ** uint256(getPoolPrecisionDecimals())` and divides the result by `self.lpToken.totalSupply()`. Even though in the current implementation inside `Swap.sol` we can see that the `LPToken` is created with the value of `SwapUtils.getPoolPrecisionDecimals()` as the number of decimals, this may change during maintenance or when adding features and there is no mechanism inside `SwapUtils.getVirtualPrice` that will indicate that the number of decimals of the `self.lpToken` is no longer equal to `getPoolPrecisionDecimals()`. This would lead to an incorrect virtual price.

Recommendation: Replace the call to `getPoolPrecisionDecimals()` inside `SwapUtils.getVirtualPrice` with a computation based on (function of) `self.lpToken.decimals()`. This will make the code future-proof.

QSP-5 Increased loss of precision due to multiplication after division

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `SwapUtils.sol`

Description: The accuracy of the return value of the `SwapUtils.getD(uint256[] memory xp, uint256 _A)` function could be affected by the loss of precision that occurs with the repeated divisions that occur on L241 (inside the `for`-loop), which are all performed before the multiplications in the subsequent iterations of the loop and the multiplication of `dP` with `numTokens` on L244 (after the `for`-loop).

```
L240: for (uint j = 0; j < numTokens; j++) {
L241:   dP = dP.mul(D).div(xp[j].mul(numTokens).add(1));
L242: }
L243: prevD = D;
L244: D = nA.mul(s).add(dP.mul(numTokens)).mul(D).div(
L245:     nA.sub(1).mul(D).add(numTokens.add(1).mul(dP)));
```

Recommendation: Use another local variable inside the `for`-loop to store the denominator of the computation separately from the numerator. The existing local variable `dP` can store the numerator and the new variable (let's call it `denom`) can store the denominator. The new code could look like the following code snippet:

```
uint256 dP = D;
uint256 denom = 1;
for (uint j = 0; j < numTokens; j++) {
  dP = dP.mul(D);
  denom = denom.mul(xp[j].mul(numTokens).add(1));
}
prevD = D;
D = nA.mul(s).add(dP.mul(numTokens).div(denom)).mul(D).div(
    nA.sub(1).mul(D).add(numTokens.add(1).mul(dP).div(denom)));
```

Update: This issue was acknowledged by adding the following comment inside the loop, after the aforementioned division and multiplication: "If we were to protect the division loss we would have to keep the denominator separate and divide at the end. However this leads to overflow with large numTokens or/and D. dP = dP * D * D * D * ... overflow!"

QSP-6 Crude check of contract address does not work

Severity: *Low Risk*

Status: Fixed

File(s) affected: `Swap.sol`

Description: The return value of the following call on L94: `allowlist.getAllowedAmount(address(this), address(0)); // crude check of the allowlist`, is ignored inside the `Swap.constructor`. It seems that this function is called to check if the `allowlist` address is indeed the address of an `Allowlist` contract instance. However, there are 2 problems with this approach:

1. The address can point to any address even `address(0)`, an EOA or a contract that does not have the `getAllowedAmount` and that call will still not cause a revert and will only return `0`.
2. If the address indeed points to the `Allowlist` contract, then the parameters passed to this call: `allowlist.getAllowedAmount(address(this), address(0));` should correctly return `0`.

Therefore, wrapping the call to `allowlist.getAllowedAmount` in a `require` statement that checks if the return value of that function is equal to `0` will not confirm if the address indeed points to the `Allowlist` contract.

Recommendation: Make sure the `Allowlist` contract instance has set the account limits and multipliers or pool caps before calling the `Swap.constructor` and then call either `allowlist.getAllowedAmount` or `allowlist.getPoolCap` with input parameters that you know will return a non-zero value. Wrap this call in a `require` statement that checks the expected (non-zero) return value.

QSP-7 Checks-effects-interactions pattern violated in `addLiquidity`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `SwapUtils.sol`

Description: The checks-effects-interactions pattern is respected by most methods with one notable exception being the `SwapUtils.addLiquidity` method.

Recommendation: We recommend calling `updateWithdrawFee` and `mint` before `safeTransferFrom` inside `addLiquidity`.

QSP-8 Privileged Roles and Ownership

Severity: *Informational*

Status: Acknowledged

File(s) affected: `Allowlist.sol`, `LPToken.sol`, `Swap.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. The following instances were identified in this project:

1. The owner of the `Allowlist` contract can perform the following actions:
   . Set the multipliers for any addresses to any value, any number of times.
   . Set the account limits for any pool to any value, any number of times.
   . Set the pool cap for any pool to any value, any number of times.
2. The owner of the `LPToken` contract can mint any amount of tokens to any address. There is no cap.
3. The owner of the `Swap` contract can:
   . Set admin fee values and withdraw the admin fees at any point in time as many times as they want.
   . Set the swap fee values at any point in time as many times as they want.
   . Set the default withdrawal fee values at any point in time as many times as they want.
   . Set the guarded status of the deposits. If set to `false` the pool will allow deposits over the allowed limit per user and will allow the TVL to go over the pool cap.

Recommendation: These centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

Update from the dev team: Our website will contain a risks page as user facing documentation that outlines privileged roles and capabilities.

QSP-9 Unlocked Pragma

Severity: *Informational*

Status: Fixed

File(s) affected: `all`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.5.*`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version.

QSP-10 Methods with different names and same implementation

Severity: *Informational*

Status: Fixed

File(s) affected: `SwapUtils.sol`

Description: The 2 functions `SwapUtils.calculateRemoveLiquidity` and `SwapUtils.calculateRebalanceAmounts` have the same implementation logic. However, they have different names and the former is `external` and the latter is `internal`.

Recommendation: Keep only the `external` method.

QSP-11 Missing assertion in `removeLiquidityImbalance`

Severity: *Informational*

Status: Fixed

File(s) affected: `SwapUtils.sol`

Description: The reference implementation of the `removeLiquidityImbalance` function contains an additional assertion that check if the `tokenAmount` is different from zero, before adding one to it. This is missing on L759 in Saddle.

Recommendation: Add the missing assertion, which would result in the following code snippet:

```
uint256 tokenAmount = D0.sub(D2).mul(tokenSupply).div(D0);
assert(tokenAmount != 0, "Burnt amount cannot be zero");
tokenAmount = tokenAmount.add(1);
```

QSP-12 Block Timestamp Manipulation

Severity: *Informational*

Status: Acknowledged

File(s) affected: `Swap.sol`, `SwapUtils.sol`

Description: Projects may rely on block timestamps for various purposes. However, it's important to realize that miners individually set the timestamp of a block, and attackers may be able to manipulate timestamps for their own purposes. If a smart contract relies on a timestamp, it must take this into account. The following functions/modifiers use the block timestamp:

- `Swap.deadlineCheck`
- `SwapUtils.calculateCurrentWithdrawFee`
- `SwapUtils.updateUserWithdrawFee`
- `SwapUtils._getAPrecise`
- `SwapUtils.rampA`
- `SwapUtils.stopRampA`

Recommendation: Warn end-users that the timestamps for deadlines can have an error of up to 900 seconds. Ensure that the withdraw fee is not severely affected by a 900 second error.

Update from the dev team: Our website will contain a risks page as user facing documentation that outlines the issue with the timestamp accuracy.

QSP-13 Accidental overwriting of multipliers

Severity: *Informational*

Status: Acknowledged

File(s) affected: `Allowlist.sol`

Description: In function `Allowlist.setMultipliers`, duplicate addresses in the input arrays will cause overwrites of multipliers (in the case of human errors), during execution.

Recommendation: Check that the input addresses are unique and have not been set before. Consider adding a Boolean flag that allows/prevents overwriting existing multipliers.

Update from the dev team: We will be communicating the multipliers to users and let users confirm the amounts. We will proceed with caution and check for duplicate addresses before calling the function.

QSP-14 Allowed amount could be higher than pool cap

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `Allowlist.sol`

Description: There is no check that constrains the owner of the `Allowlist` contract from setting the `multipliers`, `poolCaps` and `accountLimits` to such values which would result in a call to `getAllowedAmount` for some pool to be greater than `getPoolCap` for the same pool. Would this be acceptable?
Similarly the values of `multipliers` can be greater than the value of `DENOMINATOR`. Would this be acceptable?

Recommendation: Clarify what the constraints should be on the values of `multipliers`, `poolCaps` and `accountLimits`. Add the corresponding `require` statements to enforce these constraints in the setter methods.

Update from the dev team: Currently we are still finishing up on the process of how the multipliers should be determined and updated. multipliers could be higher than DENOMINATOR and this is intentional.

QSP-15 The value of the A parameter can be influenced by `block.timestamp`

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `SwapUtils.sol`

Description: Projects may rely on block timestamps for various purposes. However, it's important to realize that miners individually set the timestamp of a block, and attackers may be able to manipulate timestamps for their own purposes. If a smart contract relies on a timestamp, it must take this into account. The following functions which are used to determine the value of the A parameter use the block timestamp:

- `SwapUtils._getAPrecise`
- `SwapUtils.rampA`

- `SwapUtils.stopRampA`

The value of A is used to compute a large number of other crucial amounts in the system, including the values of D, the virtual price and the amount of tokens added/withdrawn. It is important to note that a malicious miner can change the value of `block.timestamp` with up to 900 seconds.

Recommendation: Ensure that a manipulation of the current `block.timestamp` of up to (plus/minus) 900 seconds does not affect the values of D, the virtual price and the amount of tokens added/withdrawn. This can be proven by developing unit tests to check these values when the timestamp is intentionally changed/manipulated.

Update: The dev team has added multiple test cases in their test suite, which show that the benefits for the attacker in case the timestamp would be changed by 900 seconds exists, but is not significant.

QSP-16 `stopRampA` may be called repeatedly even after a ramp has ended

Severity: *Low Risk*

Status: Fixed

File(s) affected: `SwapUtils.sol`

Description: There is no verification when the `stopRampA()` function is called to check if:

1. the current ramp has already ended due to the `self.futureATime` being in the past.
2. the `stopRampA()` was already called before to stop the current ramp.
   This allows calling this function multiple times (consecutively) with the effect that each time the `self.initialATime` is set to the current block timestamp, which prevents a new ramp for `MIN_RAMP_TIME` seconds. This could be problematic if the `stopRampA` function is called multiple times by mistake.

Recommendation: Require that `self.futureATime` not be in the past at the beginning of `stopRampA()`, that is: `self.futureATime > block.timestamp`.

QSP-17 Ramping can be started while previous ramp is still ongoing

Severity: *Informational*

Status: Fixed

File(s) affected: `SwapUtils.sol`

Description: The error message of the first `require` statement inside `rampA()` says that: "Ramp already ongoing". However, this `require` statement only checks if the `MIN_RAMP_TIME` has passed since the last call to `rampA()`, when the value of `self.initialATime` was set. The time when the previous ramp ends is actually given by `self.futureATime`, which is required to be greater or equal to `self.initialATime + MIN_RAMP_TIME` on L922.

Recommendation: Either change the error message to indicate that a new ramp cannot be started until the `MIN_RAMP_TIME` has passed from when the previous ramp was started, or change the condition in the `require` statement on L921 such that the current block timestamp is greater than `self.futureATime`.

Update: This issue was fixed by changing the error message of the `require` statement, which means that ramping can be started 1 day after the previous ramp was started, even if the previous ramp is not finished.

QSP-18 Integer Overflow

Severity: *Informational*

Status: Fixed

File(s) affected: `SwapUtils.sol`

Description: The `SwapUtils.rampA()` function uses primitive addition (+) and multiplication (*) operators on L921, L922 and L925. The latter could cause an overflow if the value of the `futureA_` input parameter is too large.

Recommendation: Use the corresponding `SafeMath` functions instead of primitive arithmetic operators.

QSP-19 Loss-Making Updates to A

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `SwapUtils.sol`

Description: This economic attack on the Curve contracts was discovered by Peter Zeitz. Since the Saddle Finance contracts are a Solidity implementation of the Curve contracts they are also vulnerable to the same attack.

Recommendation: The recommendation provided in the article linked in the description is to reduce the step size in A to no more than 0.1% per block.

Update: The dev team has implemented several test cases that indicate the effects of this attack while ramping the value of A upwards and downwards. These tests show that the only cases where the attack is successful is when the change in A is large, which may only happen if there would be 2 weeks between 2 transactions on the target pool. The dev team also indicated that they will be using 2-4 weeks as the standard ramp time to lower the risks for LPs.

Adherence to Specification

The original StableSwap paper provides the StableSwap invariant on page 5 as:
$$An^n \sum x_i + D = ADn^n + \frac{D^{n+1}}{n^n \prod x_i}$$
One can subtract $An^n \sum x_i$ from both sides of this relation to obtain: $D = ADn^n + \frac{D^{n+1}}{n^n \prod x_i} - An^n \sum x_i$
The function `SwapUtils.getD` indicates in its `@notice` comment that: "Get D, the StableSwap invariant, based on a set of balances and a particular A". However, the implemented relation looks different from the above. We are not able to understand how this relation is derived from the relation in the original StableSwap paper, mentioned at the beginning of this description. However, with the exception of one bug which we have indicated in the findings above, it is in-line with the `SwapTemplateBase.vy` implementation, which the Saddle dev team has indicated as being the reference for this audit.
We have found the following functions in Saddle, which are missing in Curve:

1. `calculateCurrentWithdrawFee` seems to calculate an additional (user specific) withdraw fee, which is unused (set to zero) in Curve.fi. Note that this fee is applied to all 3 withdrawal methods in Saddle, namely: `removeLiquidity`, `removeLiquidityOneToken` and `removeLiquidityImbalance`.

2. `updateUserWithdrawFee` which updates the `withdrawFeeMultiplier` per user and is only called by `addLiquidity`. The formula implemented inside this function is complex and we did not have any specification to compare it against. We recommend adding a comment that would indicate the desired formula for the multiplier.

Code Documentation

1. **[Fixed]** Each function should have a comment specifying its purpose, any input parameter(s) and return value(s) at the very least. The following functions do not have such comments:
    . `CERC20Utils.getUnderlyingBalances`
    . `LPToken.constructor`
    . `LPToken.mint`
    . `MathUtils.within1` we assume this returns `true` if the absolute value of the difference is less or equal to 1, i.e. `|a-b| <= 1`. Otherwise, it returns `false`.
    . `MathUtils.difference` we assume this returns the absolute value of the difference of its input parameters, i.e. `|a-b|`.
    . All functions in the `StakeableTokenWrapper` contract.

2. **[Fixed]** Function comments do not make consistent use of available and necessary NatSpec tags. Some functions only have one sentence in the `@notice` tag which mentions both the return values and parameter(s) very briefly. Other functions like the 2 overloadings of `SwapUtils._xp` on L294 and L304 have a `@notice` and a `@return` tag. Other functions, such as `Swap.constructor` have only `@param` tags. Other functions like `Swap.getToken` have a `@notice` tag and a `@param` tag but no `@return` tag.
3. **[Fixed]** The comments of the 2 overloadings of `SwapUtils._xp` on L294 and L304 have identical comments. The comment of the function on L294 needs to be updated because it does not use "the pool balances". It uses the `_balances` input parameter instead.
4. **[Fixed]** The `POOL_PRECISION` constant is mentioned in the comments listed below. However, there is no such constant. The only other constant that has a similar name is `POOL_PRECISION_DECIMALS`:
    . L46 in `Swap.sol`: "Cannot be larger than POOL_PRECISION"
    . L144 in `Swap.sol`: "@return the virtual price, scaled to the POOL_PRECISION"
    . L38 in `SwapUtils.sol`: "multipliers for each pooled token's precision to get to POOL_PRECISION"
    . L310 in `SwapUtils.sol`: "@return the virtual price, scaled to the POOL_PRECISION".
5. **[Fixed]** On L85, L172, L219 of `SwapUtils.sol` and L49, L112 of `Swap.sol` there is a comment that contains a typo: "the the amplification coefficient * n * (n - 1)". One of the 2 "the"s should be removed from each of those comments.
6. It should be made clear in user facing documentation that fees that will be charged if the user withdraws within 4 week of their deposit as seen in the function `calculateCurrentWithdrawFee()`
7. **[Fixed]** In the recurrence relation (L204 - L210) in the function `getYD` seems to be documented as shown in the code snippet below. However, there is no reference to any quadratic function in the StableSwap paper: `@dev This is accomplished via solving the quadratic equation`
8. **[Fixed]** Typo in comment on L18 in `Swap.sol`: "happen" -> "happens".
9. **[Fixed]** Multiple similar typos on L2148, L2542 in `test/swap.ts` the comment says: "Malicious miner skips 900 seconds", however, the timestamp is shifted by 2 weeks.
10. **[Fixed]** Typo on L267 `test/swap4tokens.ts`: "recieve" -> "receive".

Adherence to Best Practices

1. **[Fixed]** The `Allowlist` contract specifies events for setting the `poolCaps` and `accountLimits`. However, there is no event for setting `multipliers`. Is this intended? It would be more consistent to emit an event on every setter method that can be called by the owner of the contract.
2. **[Fixed]** The `MathUtils.within1` function could be simplified by reusing the code of `MathUtils.difference`. This was the implementation of the former function could be reduced to 1 line of code: `return difference(a, b) <= 1;`
3. **[Fixed]** Magic numbers should be replaced with named constants. For example, the magic number `256` appears 3 times in `SwapUtils.sol`.
4. **[Fixed]** If the same result of a function call is used multiple times, store the result in a local variable instead of calling the function multiple times, in order to save gas. For example:
    . The `SwapUtils._xp(self)` function is called several times (including inside the loop) in the `SwapUtils.calculateWithdrawOneTokenDY` function.
    . The `SwapUtils.feePerToken(self)` is called inside the loop from `SwapUtils.calculateWithdrawOneTokenDY`, however it has a constant value and should only be called once, before the loop.
    . Similarly, to the previous bullet point `SwapUtils.feePerToken(self)` is called inside the loop from `SwapUtils.addLiquidity`.
5. **[Fixed]** If any of the values in the `amounts` array, the input parameter for `SwapUtils.removeLiquidityImbalance`, are greater than the `balances` values corresponding to the same token, then the function will revert without a clear error message due to the subtraction on L743: `balances1[i] = balances1[i].sub(amounts[i]);` It is recommended to add a `require` statement with a clear error message, which checks that `amounts[i] <= self.balances[i]` for each token, at the beginning of the function.
6. **[Fixed]** The same issue as above would happen to the `SwapUtils.calculateTokenAmount` function if `deposit == false`.
7. **[Fixed]** `uint` should be changed to `uint256` or any other precision that is necessary for clarity and consistency.
8. **[Fixed]** In `SwapUtils.sol` in the last commit, the function does not have an explicit `return` statement if the `if`-statement on L417 is not entered. This makes the return value of this function is ambiguous. We recommend adding an explicit `return` statement at the end of this function.

Test Results

Test Suite Results

All tests in the test suite are passing. However, all tests involve at most 2 tokens in the pool. Therefore, L327 in `SwapUtils.sol` is never covered. We strongly recommend adding more tests with at least 3 tokens in the pool to assess the correctness of all implemented functionality.

Update: The test suite has been improved such that all statement are covered. This improvement includes over a dozen new tests, which also include tests having pools with more than 2 tokens, as well as simulations of attack attempts while the A parameter is ramping upwards and downwards.

```
Allowlist
  setPoolCap
     ✓ Emits PoolCap event (66ms)
     ✓ Reverts when non-owner tries to set the pool cap
     ✓ Sets and gets pool cap (109ms)
  setPoolAccountLimit & setMultiplier
     ✓ Emits PoolAccountLimit event
     ✓ Emits SetMultipliers event (1616ms)
     ✓ Reverts when non-owner tries to set the pool account limit
     ✓ Reverts when array lengths are different (59ms)
     ✓ Sets and gets pool account limit (1314ms)

MathUtils
  within1
     ✓ Returns true when a > b and a - b <= 1
     ✓ Returns false when a > b and a - b > 1
     ✓ Returns true when a <= b and b - a <= 1
     ✓ Returns false when a <= b and b - a > 1
     ✓ Reverts during an integer overflow
  difference
     ✓ Returns correct difference when a > b
     ✓ Returns correct difference when a <= b
     ✓ Reverts during an integer overflow

OwnerPausable
   ✓ Emits an event on pausing
   ✓ Reverts when pausing if already paused
   ✓ Reverts when a non-owner tries to pause
   ✓ Emits an event on unpausing (40ms)
   ✓ Reverts when unpausing if already unpaused
   ✓ Reverts when a non-owner tries to unpause

StakeableTokenWrapper
   ✓ Emits an event on staking (184ms)
   ✓ Emits an event on withdrawing (299ms)
   ✓ Only allows staked funds to be withdrawn (176ms)
   ✓ Returns correct staked balances (334ms)
   ✓ Returns correct total supply (173ms)

Swap
  swapStorage
    lpToken
       ✓ Returns correct lpTokenName
       ✓ Returns correct lpTokenSymbol
    A
       ✓ Returns correct A value
    fee
       ✓ Returns correct fee value
    adminFee
       ✓ Returns correct adminFee value
  getToken
     ✓ Returns correct addresses of pooled tokens
     ✓ Reverts when index is out of range
  getTokenIndex
     ✓ Returns correct token indexes
     ✓ Reverts when token address is not found
  getTokenBalance
     ✓ Returns correct balances of pooled tokens
     ✓ Reverts when index is out of range
  getA
     ✓ Returns correct value
  addLiquidity
     ✓ Reverts when contract is paused
     ✓ Succeeds with expected output amount of pool tokens (238ms)
     ✓ Succeeds with actual pool token amount being within ±0.1% range of calculated pool token (216ms)
     ✓ Succeeds with correctly updated tokenBalance after imbalanced deposit (155ms)
     ✓ Reverts when minToMint is not reached due to front running (209ms)
     ✓ Reverts when block is mined after deadline
     ✓ Emits addLiquidity event (59ms)
  removeLiquidity
     ✓ Succeeds even when contract is paused (270ms)
     ✓ Succeeds with expected return amounts of underlying tokens (257ms)
     ✓ Reverts when user tries to burn more LP tokens than they own (171ms)
     ✓ Reverts when minAmounts of underlying tokens are not reached due to front running (302ms)
     ✓ Reverts when block is mined after deadline (159ms)
     ✓ Emits removeLiquidity event (209ms)
  removeLiquidityImbalance
     ✓ Reverts when contract is paused (186ms)
     ✓ Succeeds with calculated max amount of pool token to be burned (±0.1%) (412ms)
     ✓ Reverts when user tries to burn more LP tokens than they own (176ms)
     ✓ Reverts when minAmounts of underlying tokens are not reached due to front running (606ms)
     ✓ Reverts when block is mined after deadline (184ms)
     ✓ Emits RemoveLiquidityImbalance event (306ms)
  removeLiquidityOneToken
     ✓ Reverts when contract is paused. (172ms)
     ✓ Succeeds with calculated token amount as minAmount (431ms)
     ✓ Reverts when user tries to burn more LP tokens than they own (192ms)
     ✓ Reverts when minAmount of underlying token is not reached due to front running (628ms)
     ✓ Reverts when block is mined after deadline (186ms)
     ✓ Emits RemoveLiquidityOne event (290ms)
  swap
     ✓ Reverts when contract is paused
     ✓ Succeeds with expected swap amounts (152ms)
     ✓ Reverts when minDy (minimum amount token to receive) is not reached due to front running (243ms)
     ✓ Succeeds when using lower minDy even when transaction is front-ran (244ms)
     ✓ Reverts when block is mined after deadline
     ✓ Emits TokenSwap event (85ms)
  getVirtualPrice
     ✓ Returns expected value after initial deposit
     ✓ Returns expected values after swaps (231ms)
     ✓ Returns expected values after imbalanced withdrawal (567ms)
     ✓ Value is unchanged after balanced deposits (531ms)
     ✓ Value is unchanged after balanced withdrawals (186ms)
  setSwapFee
     ✓ Emits NewSwapFee event
     ✓ Reverts when called by non-owners
     ✓ Reverts when fee is higher than the limit
     ✓ Succeeds when fee is within the limit
  setAdminFee
     ✓ Emits NewAdminFee event
     ✓ Reverts when called by non-owners
     ✓ Reverts when adminFee is higher than the limit
     ✓ Succeeds when adminFee is within the limit
  getAdminBalance
     ✓ Is always 0 when adminFee is set to 0 (134ms)
     ✓ Returns expected amounts after swaps when adminFee is higher than 0 (221ms)
  withdrawAdminFees
     ✓ Reverts when called by non-owners
     ✓ Succeeds with expected amount of fees withdrawn (244ms)
     ✓ Withdrawing admin fees has no impact on users' withdrawal (1845ms)
  Guarded launch
     ✓ Only owner can remove the guard
     ✓ Reverts when depositing over individual limit (155ms)
     ✓ Reverts when depositing over pool cap (148ms)
  Test withdrawal fees on removeLiquidity
     ✓ Removing liquidity immediately after deposit (230ms)
     ✓ Removing liquidity 2 weeks after deposit (217ms)
     ✓ Removing liquidity 4 weeks after deposit (222ms)
  Test withdrawal fees on removeLiquidityOne
     ✓ Removing liquidity immediately after deposit (562ms)
     ✓ Removing liquidity 2 weeks after deposit (365ms)
     ✓ Removing liquidity 4 weeks after deposit (370ms)
  Test withdrawal fees on removeLiquidityImbalance
     ✓ Removing liquidity immediately after deposit (285ms)
     ✓ Removing liquidity 2 weeks after deposit (290ms)
     ✓ Removing liquidity 4 weeks after deposit (284ms)
  updateUserWithdrawFee
     ✓ Test adding liquidity, and once again at 2 weeks mark then removing all deposits at 4 weeks mark (366ms)
  setDefaultWithdrawFee
     ✓ Emits NewWithdrawFee event
     ✓ Setting the withdraw fee affects past deposits as well (164ms)
     ✓ Reverts when fee is too high
  rampA
     ✓ Emits RampA event
     ✓ Succeeds to ramp upwards (66ms)
     ✓ Succeeds to ramp downwards (70ms)
     ✓ Reverts when non-owner calls it
     ✓ Reverts with 'New ramp cannot be started until 1 day has passed'
     ✓ Reverts with 'Insufficient ramp time'
     ✓ Reverts with 'futureA_ must be between 0 and MAX_A'
     ✓ Reverts with 'futureA_ is too small'
```

```
    ✓ Reverts with 'futureA_ is too large'
  stopRampA
    ✓ Emits StopRampA event
    ✓ Stop ramp succeeds (95ms)
    ✓ Reverts with 'Ramp is already stopped' (84ms)
  Check for timestamp manipulations
    ✓ Check for maximum differences in A and virtual price (115ms)
    Check for attacks while A is ramping upwards
      When tokens are priced equally: attacker creates massive imbalance prior to A change, and resolves it after
        ✓ Attack fails with 900 seconds between blocks (281ms)
        ✓ Attack fails with 2 weeks between transactions (mimics rapid A change) (256ms)
      When token price is unequal: attacker 'resolves' the imbalance prior to A change, then recreates the imbalance.
        ✓ Attack fails with 900 seconds between blocks (268ms)
        ✓ Attack succeeds with 2 weeks between transactions (mimics rapid A change) (263ms)
    Check for attacks while A is ramping downwards
      When tokens are priced equally: attacker creates massive imbalance prior to A change, and resolves it after
        ✓ Attack fails with 900 seconds between blocks (276ms)
        ✓ Attack succeeds with 2 weeks between transactions (mimics rapid A change) (268ms)
      When token price is unequal: attacker 'resolves' the imbalance prior to A change, then recreates the imbalance.
        ✓ Attack fails with 900 seconds between blocks (274ms)
        ✓ Attack fails with 2 weeks between transactions (mimics rapid A change) (279ms)

Swap with 4 tokens
  addLiquidity
    ✓ Add liquidity succeeds with pool with 4 tokens (251ms)
  swap
    ✓ Swap works between tokens with different decimals (414ms)
  removeLiquidity
    ✓ Remove Liquidity succeeds (411ms)
  Check for timestamp manipulations
    ✓ Check for maximum differences in A and virtual price (136ms)
    Check for attacks while A is ramping upwards
      When tokens are priced equally: attacker creates massive imbalance prior to A change, and resolves it after
        ✓ Attack fails with 900 seconds between blocks (305ms)
        ✓ Attack fails with 2 weeks between transactions (mimics rapid A change) (313ms)
      When token price is unequal: attacker 'resolves' the imbalance prior to A change, then recreates the imbalance.
        ✓ Attack fails with 900 seconds between blocks (347ms)
        ✓ Attack succeeds with 2 weeks between transactions (mimics rapid A change) (337ms)
    Check for attacks while A is ramping downwards
      When tokens are priced equally: attacker creates massive imbalance prior to A change, and resolves it after
        ✓ Attack fails with 900 seconds between blocks (322ms)
        ✓ Attack succeeds with 2 weeks between transactions (mimics rapid A change) (327ms)
      When token price is unequal: attacker 'resolves' the imbalance prior to A change, then recreates the imbalance.
        ✓ Attack fails with 900 seconds between blocks (339ms)
        ✓ Attack fails with 2 weeks between transactions (mimics rapid A change) (335ms)


  137 passing (3m)
```

Code Coverage

All coverage values except for the branch coverage are at high levels. However, we strongly recommend increasing all branch coverage scores up to 100% to guarantee that all functionality is automatically tested such that bugs can be discovered automatically when doing maintenance or vulnerability fixes.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	98.35	76.58	98.81	98.38	
Allowlist.sol	100	66.67	100	100	
CERC20.sol	0	0	0	0	… 27,28,29,32
LPToken.sol	100	50	100	100	
MathUtils.sol	100	100	100	100	
OwnerPausable.sol	100	100	100	100	
StakeableTokenWrapper.sol	100	50	100	100	
Swap.sol	100	70	100	100	
SwapUtils.sol	100	81.48	100	100	
All files	**98.35**	**76.58**	**98.81**	**98.38**	

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
67e9537b0492335c0a35e06b2592b381bf4a98b98cc55b274e4eb9fcf499a84d ./saddle/saddle-contract/contracts/OwnerPausable.sol
66b9c330e3a2ce83397a6bc7be859f45961ea9e297087c19643ad8813b1353b5 ./saddle/saddle-contract/contracts/CERC20.sol
669fe8749faee14649bec8827cf3d6ce4dbb20380b0ded5323e56520671daa7c ./saddle/saddle-contract/contracts/MathUtils.sol
a15195d4df8dec031e115995957c9a6b4d4b15735ff20e3353ebb6fb97e41d61 ./saddle/saddle-contract/contracts/StakeableTokenWrapper.sol
6bf0bf1fd4919b0e41d61acdabb8447bde62e7fbefbc6d8a61911d01c898810f ./saddle/saddle-contract/contracts/Allowlist.sol
13162ebae74af60302b89e12c3206e7c1a1c3d98bfc94148e5c1f059aabbfb35 ./saddle/saddle-contract/contracts/LPToken.sol
ef8d1ac76a868c3d162739df2855cd79802c66d83de04c975e307086b00d8e1c ./saddle/saddle-contract/contracts/Swap.sol
ddd00f529318417f4123fdccfbceed1239eaf0c26c41476ade5a3300a0307ab2 ./saddle/saddle-contract/contracts/SwapUtils.sol
1490962b2aa9330ac086cc2e4a818ccdb3c7d53cae7f6610ff61bfccc285a083 ./saddle/saddle-contract-master/contracts/OwnerPausable.sol
31e44b42dd7ba840d44c0f8ac4755d454326c7dca2293307f372f03c169117cc ./saddle/saddle-contract-master/contracts/CERC20.sol
01d44ba2a48ae729a55a4693edbdee2c0d0b240d9a2ede20a74e69dd332da662 ./saddle/saddle-contract-master/contracts/MathUtils.sol
722060fc84d095e3744c1923fcabcf2a61d63a382fddfeabf3e158af06ed1b5b ./saddle/saddle-contract-master/contracts/StakeableTokenWrapper.sol
6e6e637694ea40136cd3a0a29527248fa1e3c5385d25e38607b53244a3c0c3a7 ./saddle/saddle-contract-master/contracts/Allowlist.sol
9b57ca96cd16d33ca0ab4bbfce489a3b00deef3471b0fc15c0551ba0e1079429 ./saddle/saddle-contract-master/contracts/LPToken.sol
817233e89ca3c23f05443961617cbdf80ab0d5e82620b6b24d353629ff672064 ./saddle/saddle-contract-master/contracts/Swap.sol
a53135e0e541c033cfd25b4233119c4f68e3b270033b1d6e3c50cde167faaeac ./saddle/saddle-contract-master/contracts/SwapUtils.sol
```

Tests

```
d96df5ca3e4a91db14ef49df16970cc7bac9d0fea467a73eac34722205e6a485 ./test/mathUtils.ts
05f7f3b8fd924d416f236bf307a37e5605f9a704633048ac4ab5976f1d2ba89d ./test/ownerPausable.ts
6f55ec6d16f5c8494439f17872cf4ce7940cd7aa9a2e8d7d7de670d4d9d47b2a ./test/allowlist.ts
cc7eee39aae25d00513c9413008552af03d4f084dc090583ee873e334ea4af32 ./test/testUtils.ts
50abe9b4899ec323c51ebd158bdd55f725d9ded3ff991be59ab9124b188c9555 ./test/swap.ts
ddd4bceddf74ae6a314dee936720ba6b0988c0efb404ccb018b2bcea15e6f50b ./test/stakeableTokenWrapper.ts
e93831cc015f5c0b98ac20569f724d60a697b121c109d152511c5603b06c55c6 ./test/swap4tokens.ts
```

Changelog

- 2020-11-03 - Initial report based on commit `83491b3`
- 2020-11-18 - Updated report based on commit `5a56e24`
- 2020-12-25 - Updated report based on commit `ebec9fd`
- 2020-12-01 - Updated report based on commit `759c028`
- 2020-12-07 - Updated report based on commit `33baaaa`
- 2020-12-09 - Updated report based on commit `08c06c1`

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.