Quantstamp Security Assessment Certificate

March 14th 2023 — Quantstamp Verified

CrocSwap

This audit report was prepared by Quantstamp, the leader in blockchain security.

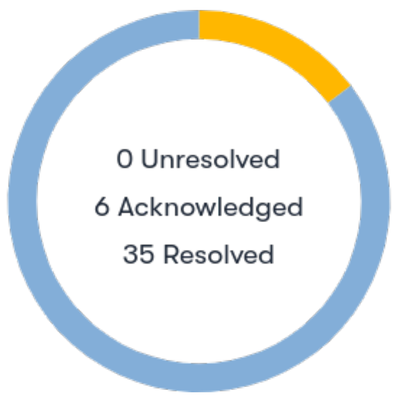
Executive Summary

Type	Automated Market Maker
Auditors	Fayçal Lalidji, Senior Security Engineer Roman Rohleder, Research Engineer Alejandro Padilla Gaeta, Research Engineer
Timeline	2022-06-27 through 2022-09-02
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	High
Test Quality	High

Source Code

Repository	Commit
CrocSwap Protocol	54ca03b

Total Issues	**41** (35 Resolved)
High Risk Issues	**2** (2 Resolved)
Medium Risk Issues	**8** (7 Resolved)
Low Risk Issues	**7** (6 Resolved)
Informational Risk Issues	**16** (12 Resolved)
Undetermined Risk Issues	**8** (8 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).
Fixed	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

Initial audit
Through reviewing the code, we found **41 potential issues.** We recommend carefully re-considering the logic to ensure the safety of users and the application design. Due to the complexity of the project Crocswap team must take extra attention when pushing the fixes. In fact, given the number of findings and the complexity of the codebase, we recommend enhancing the test suite with tests covering edge cases, negative cases, etc... to reach higher branch coverage results than the actual score that is equal to 90%.
Fix review update
All highlighted issues in the initial audit have been either fixed, mitigated, or acknowledged except "Unlocked Pragma" which remains unresolved. Please note that QSP-39 to QSP-41 were added after the fix review and are unresolved.
Second fix review update
All highlighted issues in the initial audit have been either fixed, mitigated, or acknowledged.

ID	Description	Severity	Status
QSP-1	Multipath Bypasses Reentrancy Locks and Can Allow Ether Double Spend	High	Fixed
QSP-2	Replay Attack when Resetting Nonce	High	Fixed
QSP-3	JIT Liquidity MEV Still Partially Possible	Medium	Mitigated
QSP-4	Proxy Calls Might Fail Silently	Medium	Mitigated
QSP-5	Possible to Upgrade Sidecar to Invalid Contract	Medium	Fixed
QSP-6	Incorrect Parsing of Long Form Orders	Medium	Fixed
QSP-7	Possible for Router to Bypass Restrictions	Medium	Fixed
QSP-8	`deflateLiqSeed` Is Incorrect	Medium	Fixed
QSP-9	`shaveRoundLots` Is Rounding Liquidity to 2048, Not 1024	Medium	Acknowledged
QSP-10	Checks-Effects-Interactions Pattern Violation	Medium	Mitigated
QSP-11	Critical Role Transfer Not Following Two-Step Pattern	Low	Fixed
QSP-12	`mulQ48` Is Incorrect	Low	Fixed
QSP-13	Pools Created with Invalid Protocol Take	Low	Fixed
QSP-14	Error Messages Are Not Bubbled up From Sidecars	Low	Fixed
QSP-15	Commands Executed in Sidecars Are Not Returning Values	Low	Fixed
QSP-16	Possible to Pass Malleable Signatures	Low	Acknowledged
QSP-17	Missing Underflow Checks in `LiquidityMath.sol`	Informational	Fixed
QSP-18	Missing Input Validation	Informational	Mitigated
QSP-19	Privileged Roles and Ownership	Informational	Acknowledged
QSP-20	Unlocked Pragma	Informational	Fixed
QSP-21	Upgradability Risks	Informational	Acknowledged
QSP-22	Invalid Commands Are Not Handled Properly	Informational	Fixed
QSP-23	Pools Might Be Interacting with Unsafe Tokens	Informational	Acknowledged
QSP-24	`collectEther` Should only Be Called Once	Informational	Fixed
QSP-25	`unchecked` Keyword Risk	Informational	Mitigated
QSP-26	Risk when Passing Structs	Informational	Acknowledged
QSP-27	Risk Using Constants for Command Codes	Informational	Fixed
QSP-28	`deltaQuote` Method Is Called with the Wrong Parameters	Informational	Fixed
QSP-29	`isPoolInit` Will Be Incorrect if New Schemas Are Added	Informational	Fixed
QSP-30	Use of `call` when Transferring Ether	Informational	Mitigated
QSP-31	Missing Overflow Check in `LiquidityMath.shaveRoundLotsUp()`	Undetermined	Fixed
QSP-32	Use of Unsafe Cast Operation in `Chaining.pinFlow()`	Undetermined	Fixed
QSP-33	Missing Licenses	Undetermined	Fixed
QSP-34	Several Methods Are Unclear	Undetermined	Mitigated
QSP-35	Liquidity 0 Is Allowed	Undetermined	Fixed
QSP-36	Possible to Call Different Versions of `HotPath`	Undetermined	Fixed
QSP-37	Use of Virtual Tokens Appears Unnecessary	Undetermined	Fixed
QSP-38	Virtual Deposit Can Be Exploited	Undetermined	Fixed
QSP-39	Incorrect Condition in `TradeMatcher.sweepSwapLiq()`	Low	Fixed
QSP-40	Unclear Usage of the Fallback Function	Informational	Fixed
QSP-41	Documentation Issues and Best Practices (Fix Review)	Informational	Fixed

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 is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review is a review of smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on 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 v0.8.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Multipath Bypasses Reentrancy Locks and Can Allow Ether Double Spend

Severity: *High Risk*

Status: Fixed

File(s) affected: `MultiPath.sol`, `WarmPath.sol`, `CrocSwapDex.sol`, `AgentMask.sol`

Description: The `MultiPath` sidecar bypasses controls set in CrocSwap in two ways:

- *Allows Ether double spend*: If the MultiPath sidecar is used to execute multiple commands requiring sending Ether, the system will not realize that it's spending the same Ether more than once. For example, the user can mint positions that require sending Ether, but will only be charged for the Ether once (the same can happen with swap). This will easily lead to funds lost.
- *Bypasses reentrancy locks*: The methods in the `CrocSwapDex` are annotated with modifiers that prevent the method from being called again to protect them from reentrancy. However, this is the only place with that kind of protection; all commands called within do not check for reentrancy. Normally this is not a problem, however, the `MultiPath` sidecar allows calling an arbitrary number of user commands in any order, thereby bypassing the reentrancy locks.

Recommendation: Remove the `MultiPath` sidecar. While executing several commands in a single call can be convenient, it unnecessarily increases the attack surface. Moreover, similar behavior can be achieved in a safer way through long-form orders or external smart contracts. Even if `MultiPath` sidecar is removed, future code upgrades could introduce reentrancy. Be careful always to follow the checks effects interaction pattern and to make proper use of the reentrancy locks.

Update: Fixed in commit 26c2496 by removing the `MultiPath` sidecar, as suggested.

QSP-2 Replay Attack when Resetting Nonce

Severity: *High Risk*

Status: Fixed

File(s) affected: `AgentMask.sol`, `ColdPath.sol`

Description: `AgentMask.resetNonce` and `AgentMask.resetNonceCond` allow the user to reset its nonce value. This issue may allow an attacker to execute a replay attack if the nonce is reset to a preceding value.

Recommendation: It should be clearly documented why users are allowed to reset their nonce value.

Update: Fixed in commit b013653. Please be advised that the updated fix will disallow the execution of any signed messages having a nonce value that is either greater than the previous value or lower than the updated value.

QSP-3 JIT Liquidity MEV Still Partially Possible

Severity: *Medium Risk*

Status: Mitigated

File(s) affected: `PoolRegistry.sol`, `PositionRegistry.sol`

Description: In order to mitigate JIT liquidity, a form of 'Maximal Extractable Value' (MEV), a time-locking mechanism was introduced (`jitThresh_`, `PoolRegistry.assertJitSafe()`), allowing one to close ones concentrated liquidity position only after a certain set time. This mitigation prevents executing JIT liquidity investments from being placed and removed within the same block and thereby prevents JIT liquidity actors relying on flashloans.
However, since this timer is only enforced on the burn-/withdraw-side, it still allows for non-flashloan based JIT liquidity investments.

Exploit Scenario: 1. JIT trader sees a pending swap transaction that has not yet been committed on the chain.

1. Trader mints a large amount of concentrated liquidity into the pool at said range.
2. Trader collects a large amount of fees due to this trade.
3. Trader withdraws his position as soon as `jitThresh_` runs out to suffer minimal capital risk. (See "JIT Liquidity" in the CrocSwap documentation.)

Recommendation: Consider adding another time-locking mechanism for when adding concentrated liquidity (mint-side), to mitigate this angle of MEV as well, or moving the existing time-locking from the burn-side to the mint-side.

Update: Mitigated in commit 6aa94dd by adding further documentation regarding the issue and its potential impact.

QSP-4 Proxy Calls Might Fail Silently

Severity: *Medium Risk*

Status: Mitigated

File(s) affected: `ColdInjector.sol`

Description: Before delegating a call to a proxy sidecar, the `ColdInjector` calls the `assertProxy` method (`L52`) to validate that the proxy address is not `address(0)`. However, this is insufficient; the address could still refer to an empty contract or to an externally owned account (EOA). If this ever happened, `delegatecall` would return `true` without executing anything; this could be especially dangerous if the command is an important call, like a sudo protocol command.

Recommendation: Refactor the `assertProxy` method to also validate that the proxy is a valid contract (check OpenZeppelin's `Address.isContract` as reference). If this is considered too gas-intensive, at least validate that every proxy has a valid contract address at creation time and every time a proxy is upgraded (using `upgradeProxy`).

Update: Mitigated in commit d4405c9 by adding upgrade-time checks to verify the target address to be a contract.

QSP-5 Possible to Upgrade Sidecar to Invalid Contract

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `ColdPath.sol`

Description: The `upgradeProxy` method (`L223`) does not validate that the new proxy address is correct. For example,

- The new proxy address might be the same as before (in other words, no update happened).
- The new proxy address could be `address(0)`.
- The new address might not be a valid contract (it could be an empty contract or an externally owned account).
- The new address might not be the right proxy contract.

This could be particularly dangerous if the `ColdPath` is the one being updated; a wrong update could leave the system unable to be upgraded.

Recommendation: If possible, consider moving the upgrade logic to the `CrocSwapDex` contract itself (this call would need to be properly protected - be very careful with this kind of change).
Also, refactor the `upgradeProxy` method to validate that the address is a valid contract.
To make the process even safer, before completing the upgrade, the proxy could "accept" the upgrade (see Compound's implementation as an example).

Update: Fixed in commit 7bc2e39 by adding explicit 2-step role/proxy acceptance functions.

QSP-6 Incorrect Parsing of Long Form Orders

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `Encoding.sol`

Description: Long-form orders are passed as a set of bytes encoded according to the rules described in this document. When an order is received, the `Encoding.sol` contract is called to decode each of the fields in the order. Unfortunately, the functionality in the `Encoding` contract used for parsing signed integers (`eatInt*` methods) is incorrect. These methods are supposed to read the first byte in the input to determine if the number is negative and the remaining 31 bytes for the actual magnitude, for a total of 32 bytes. Instead, the methods are reading the first byte (moving the offset 1 byte) and reading the next 32 bytes, for a total of 33 bytes. Not only does this return the wrong value for the number read, but it also skews any field read after.

Recommendation: Consider encoding and decoding signed integers using Solidity's abi. Also, make sure to add unit tests to ensure that all the encoding methods are working properly across all cases.

Update: Fixed in commit 6edaf9a by replacing the custom parser functions with the built-in `abi.decode()` function, as suggested.

QSP-7 Possible for Router to Bypass Restrictions

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `AgentMask.sol`

Description: CrocSwap allows users to register a router to execute a certain number of commands on their behalf. Unfortunately, there's no restriction on the types of commands the router can execute, so the router can easily abuse this. For example, if the user enables a router to execute one call on their behalf, the router could call the `approveRouter` command to give itself the right to execute an unlimited number of calls, thereby getting full control of everything the user can do.

Recommendation: Limit the types of commands that routers can execute on users' behalf.

Update: Fixed in commit 787efdb by restricting routers from calling cold path-related functions (administrative functions).

QSP-8 `deflateLiqSeed` Is Incorrect

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `CompoundMath.sol`

Description: Inside the `deflateLiqSeed` method (`L155`), the `liq` parameter is given as a `uint128` and then shifted left 48 positions, without casting it first to a `uint256`. Because of this, any information stored in the 48 top-most bits will be lost (the result is stored as a `uint256`, but at that point, the damage is already done).

Recommendation: Before shifting left, cast the `liq` parameter to `uint256`. Also, make sure to add unit tests to validate that the method works for all cases (consider edge cases).

Update: Fixed in commit 588a16c by casting `liq` to `uint256` before shifting, as suggested.

QSP-9 `shaveRoundLots` Is Rounding Liquidity to 2048, Not 1024

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `LiquidityMath.sol`

Description: The `shaveRoundLots` (`L100`) and `shaveRoundLotsUp` (`L109`) methods are supposed to truncate an amount of liquidity into a quantity that is a multiple of 1024. To do so, they are zeroing the first bits of the number. However, instead of zeroing 10 bits (`2^10 == 1024`), the methods are zeroing 11 bits. Therefore, they are truncating to multiples of 2048, not 1024.

Recommendation: Update the `LOT_ACTIVE_BITS` to 10. Also, add unit tests to validate that the methods are truncating to multiples of 1024 as expected.

Update: This behaviour was acknowledged by the developers to be intentional and corresponding inline code comments have been updated accordingly in commit 050376f.

QSP-10 Checks-Effects-Interactions Pattern Violation

Severity: *Medium Risk*

Status: Mitigated

File(s) affected: `DepositDesk.sol`, `SettleLayer.sol`

Related Issue(s): SWC-107

Description: The Checks-Effects-Interactions coding pattern is meant to mitigate any chance of other contracts manipulating the state of the blockchain in unexpected and possibly malicious ways before control is returned to the original contract. As the name implies, external calls or interactions with other contracts should be done only after checking that the appropriate conditions are met and that the internal state is properly updated.

- `DepositDesk.disburseSurplus()` modifies `userBals_[key].surplusCollateral_` after executing an external call.
- `SettleLayer.settleFinal()` calls `settleLeg()` that executes an external call (ether or token transfer) before writing the complete state to the chain .

Recommendation: We recommend refactoring the code so that it conforms to the Checks-Effects-Interactions pattern.

Update: Partially fixed (`DepositDesk.disburseSurplus()`) in commit 061d713.

- `DepositDesk.disburseSurplus()`: Fixed.
- `SettleLayer.settleFinal()`: **Unresolved**.

QSP-11 Critical Role Transfer Not Following Two-Step Pattern

Severity: *Low Risk*

Status: Fixed

File(s) affected: `CrocPolicy.sol`

Description: The treasury privilege can be transferred to another address simply by calling the `CrocPolicy.transferGovernance()` function. This function immediately transfers a high level privilege to new addresses in a single transaction, which can be risky from a security perspective, as providing a faulty address may lock out that role from future calls.
A more secure pattern for such privilege transfers is to require the new pending addresses to issue an `acceptAdmin()` function call before finalizing the transfer. Note that this pattern is common even with the use of timelocks, such as the Compound Timelock contract.

Recommendation: Make sure that before transferring an authority the new account makes a call to the `acceptAdmin()` method to accept the role. Consider using the [Compound Timelock]() as reference.

Update: Fixed in commit 9db85f2 by making use of a two-step role transfer pattern, as suggested.

QSP-12 `mulQ48` Is Incorrect

Severity: *Low Risk*

Status: Fixed

File(s) affected: `FixedPoint.sol`

Description: The `mulQ48` is used to multiply a `Q64.64` number by a `Q16.48` (stored as `uint128` and `uint64`, respectively). The method calculates the result correctly, a number 144 bits in size (the result type is `uint144`). However, just before returning, the method casts the number down to `uint128`. This means that if the number is bigger than 128 bits, the 16 most significant bits will be silently lost.

Recommendation: Instead of casting the number to `uint128`, cast it to `uint144` so that it matches the return type. Also, add unit tests (for both normal and edge cases) to ensure the method returns the right value.

Update: Fixed in commit 58c96c3 by changing the cast from `uint128` to `uint144`, as suggested.

QSP-13 Pools Created with Invalid Protocol Take

Severity: *Low Risk*

Status: Fixed

File(s) affected: `PoolRegistry,sol`

Description: When a pool is created using the `registerPool` method, the pool is written to storage in `L219`. However, the `protocolTake` is updated after (on `L220`). Therefore, the `protocolTake_` will stay uninitialized; it will always be 0.

Recommendation: Update the `registerPool` method to set the `protocolTake_` before writing the pool into storage. Also, add a unit test to validate that the right `protocolTake_` (and other parameters) is set when initializing a pool.

Update: Fixed in commit 9e6c01f by setting `protocolTake_` before writing the pool into storage, as suggested.

QSP-14 Error Messages Are Not Bubbled up From Sidecars

Severity: *Low Risk*

Status: Fixed

File(s) affected: `ColdInjector.sol`

Description: The methods using `delegatecall` to call sidecars are only checking if the operation itself was successful or not (e.g., `callUserCmd:L39`). When a call is not successful, the method reverts without any error message; the caller won't have any information about why it failed.

Recommendation: Refactor the code to bubble up the reason why a `delegatecall` method failed (consider using OpenZeppelin's `Address.verifyCallResult` as a reference).

Update: Fixed in commit 9f4ec5e by bubbling up error messages.

QSP-15 Commands Executed in Sidecars Are Not Returning Values

Severity: *Low Risk*

Status: Fixed

File(s) affected: `HotPath.sol`, `ColdPath.sol`, `LongPath.sol`, `MultiPath.sol`, `SafeModePath.sol`,

Description: Several sidecars in the codebase do not return any value. While `ColdPath` and `SafeModePath` might not be returning anything on purpose, the `HotPath`, `LongPath` and `MultiPath` should be returning the resulting flows.

Recommendation: Make sure that all the call paths return the right values and add unit tests to validate that this is always the case.

Update: Fixed in commit eb05128 by returning values for `userCmd()` in contracts `LongPath` and `HotProxy`.

QSP-16 Possible to Pass Malleable Signatures

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `AgentMask.sol`

Description: The function `verifySignature` is prone to signature malleability since it does not validate the value of `s`.

Recommendation: We recommend adding the validation of `s`. Consider using OpenZeppelin's implementation of `tryRecover` in ECDSA.sol as reference.

Update: Acknowledged by the developers with:

> We acknowledge the signature is malleable, but believe it poses no risk because the signature image itself is not being used for any sort of indexing or replay protection. In the worse case an attacker could malform the signature, but it would have the exact same effect as an honest signature. Replay would not be possible, because both the honest and malformed signature would have the same content and hence same nonce.

Additional inline documentation was added in commit f60ad8d.

QSP-17 Missing Underflow Checks in `LiquidityMath.sol`

Severity: *Informational*

Status: Fixed

File(s) affected: `LiquidityMath.sol`

Description: The comment on `L43` (`LiquidityMath.sol`) states that the function `minusDelta()` (and `minusLots()`) will revert if the subtraction underflows. However, the code does not perform any explicit checks. If the file is ever compiled on a Solidity version below `0.8.0` (as allowed by the `pragma solidity >=0.5.0` statement on `L2`), those underflows will go unnoticed and not revert.
Similarly, the additions `(termX + 1) + (termY + 1)` on `L162` should be checked against overflows.
While in the given project all files using this library require at least solidity version `0.8.4`, if this library is re-used in another project or code in this project changes, underflows may occur.

Recommendation: We recommend adding `require(...)` statements in functions `minusDelta()` and `minusLots()`, as for example is done in `addLiq()`, to check for underflows and conform to the code comment.

Update: Fixed in commit 3e5c5ea by changing the solidity version pragma in `LiquidityMath.sol` to `^0.8.4`, which performs implicit overflow/underflow checks.

QSP-18 Missing Input Validation

Severity: *Informational*

Status: Mitigated

File(s) affected: `CrocPolicy.sol`, `ColdPath.sol`

Description: There are several methods across the codebase that lack validation. The following list contains the cases that we found (list might not be exhaustive):

1. `CrocPolicy.sol:declareAuthority(L105)` - Possible to upgrade the different authorities (ops, treasury, and emergency) to zero addresses. Although it's unlikely that governance would let an invalid address through, these are extremely important accounts, so it's better to be extra safe.
2. `ColdPath.sol:setNewPoolLiq(L174)` - Possible to set the liquidity to an impractically large or small value. This could affect how easy or hard it is to create a new pool.
3. `ColdPath.sol:setRelayerTakeRate(L166)`, `setTakeRate(L158)`, `resyncTakeRate(L182)` - There's no validation on the protocol take rates. The value is from 0 to 255. If the value is 255, the protocol would take almost the whole fee.
4. `ColdPath.sol:collectProtocol(L245)` - The method allows transferring the protocol fees to any address passed as a parameter (even `address(0)`). If the governance was ever compromised, this would allow the stealing of those funds. **Recommendation:** Consider transferring the funds to an address previously set by governance (any changes to it should be done in a timelock to give enough time for everyone to review it).
5. `PoolRegistry.sol:setPoolTemplate(L130)` - If the template is permissioned, there is no validation to ensure that the leading 160 bits of the `poolIdx` actually point to a valid permission oracle.
6. `CrocSwapDex.sol`: `constructor()` (`L43`) does not validate `authority` and `coldPath` addresses.
7. `ColdPath.sol`: `transferAuthority()` (`L252`) does not validate the authority address.
8. `TransferHelper.sol`: `safeEtherSend()` (`L42`) does not check that parameter `to` is different from `address(0)`.
9. `ColdPathsol`: `upgradeProxy()` (`L223`) does not check that parameter `proxy` is different from `address(0)`.

Recommendation: Consider refactoring the code with additional validation. Also, add unit tests to ensure that the system always deals gracefully with both, correct and incorrect values.

Update: Mitigated in commit 4ac03ee by fixing all but one findings.

1. Fixed (Fixed by fix of QSP-11).
2. Fixed (Limiting the new minimum pool liquidity to `uint constant MAX_INIT_POOL_LIQ = 10_000_000`).
3. Fixed (Limiting the new relayer take rate to `uint8 MAX_TAKE_RATE = 128`).
4. Fixed (Locking the receiver address to always be `treasury_`).
5. Fixed (Checking the resulting oracle address to be non-zero and implementing `acceptsPermitOracle()`).
6. Fixed (Now deriving said variables implicitly).
7. Fixed (Checking the resulting oracle address to be non-zero and implementing `acceptsCrocAuthority()`).
8. **Unresolved**.
9. Fixed (Contract refactored to `BootPath.sol` and now validating `proxy`).

QSP-19 Privileged Roles and Ownership

Severity: *Informational*

Status: Acknowledged

File(s) affected: `CrocPolicy.sol`

Description: Certain contracts have state variables, e.g. `owner`, which provide certain addresses with privileged roles. Such roles may pose a risk to end-users.
The following list describes the roles introduced in the `CrocPolicy` contract and the privileged operations they can perform:

- `opsAuth`, as initialized during the `constructor()` call:
  - Call arbitrary non-privileged commands (`protocolCmd()`), by calling `opsResolution()`.
  - Add/Overwrite expired policies with a new one (Whitelist an arbitrary address for executing a command), by calling `setPolicy()`.
- `emergencyAuth`, as initialized during the `constructor()` call:
  - Call all the above mentioned actions by `opsAuth`.
  - Disabling all `HotPath` functionality and entering safe mode, by calling `emergencyHalt()`.
  - Reset policy rules even before passing the corresponding mandate time, by calling `emergencyReset()`.

- `treasuryAuth`, as initialized during the `constructor()` call:
  . Call all the above mentioned actions by `opsAuth`.
  . Assign arbitrary new addresses for the roles `opsAuth`, `treasuryAuth` and `emergencyAuth`, by calling `transferGovernance()`.
  . Call arbitrary privileged commands (`protocolCmd()`), by calling `treasuryResolution()`.
  . Add/Overwrite existing policies with a new one (Whitelist an arbitrary address for executing a command), by calling `forcePolicy()`.

Recommendation: The different types of CrocSwap authorities have significant power over pools and contracts. They can pause swaps (in case of an emergency), increase or lower fees, and upgrade contracts. Each of these operations requires trust and entails risks that the users must be aware of. Clearly inform users about CrocSwap governance special rights over pools and contracts so that they are aware of any potential risks and clarify the impact of these privileged actions to the end-users via publicly facing documentation.

Update: Acknowledged by the developers and added additional documentation (`GovernanceRoles.md`) concerning privileged roles.

QSP-20 Unlocked Pragma

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/*`

Description: Every Solidity file specifies in the header a version number of the format pragma solidity (^)0.8.* . 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". Similarly, a preceding greater-or-equal (>=) allows for the specified version and above to be used.

Recommendation: For consistency and to prevent unexpected behavior in the future, removing the caret or greater-or-equal signs is recommended to lock the file onto a specific Solidity version.

Update: Fixed in commit 7784d34.

QSP-21 Upgradability Risks

Severity: *Informational*

Status: Acknowledged

File(s) affected: `contracts/*`

Description: While upgradability is not a vulnerability in itself, users should be aware that CrocSwap governance has the right to upgrade the vast majority of the codebase (all call paths and related code) at any given time. This audit does not guarantee the behavior of any future upgraded code. Also, it's important to remember that if call paths are upgraded independently there can be compatibility issues between their code (there's a lot of code shared between them). THIS COULD CAUSE SEVERE UNEXPECTED CONSEQUENCES. BE EXTREMELY CAREFUL WHEN PERFORMING UPGRADES!

Recommendation: CrocSwap users must be clearly informed that the codebase can be upgraded at any time and that there are risks involved with them. Also, be extremely careful when performing upgrades.

Update: Acknowledged by the developers and added additional documentation (`UpgradeGuidelines.md`) concerning upgradeability.

QSP-22 Invalid Commands Are Not Handled Properly

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/callpaths/*`

Description: The `userCmd` and `protocolCmd` methods in all call paths receive a command code as the first parameter (encoded as bytes). In most call paths, this code goes through a set of `if/else` statements to determine which command to execute. However, no call path is handling explicitly an invalid command code; they will continue executing the line of code after the `if/else` statement. At the moment, this does not appear to cause big problems (only `settleFlows` is being called in the `KnockoutPath.sol` and `WarmPath.sol` but with empty flows). However, this could become a big problem in future versions of the code.

Recommendation: Make sure that all call paths revert if they receive a code that does not match any of the valid commands. Also, add unit tests to ensure that all call paths revert if given an invalid code.

Update: Fixed in commit 6c7bec2 by reverting in `userCmd()` of `ColdPath.sol` and `BootPath.sol`, when an unknown command is provided.

QSP-23 Pools Might Be Interacting with Unsafe Tokens

Severity: *Informational*

Status: Acknowledged

File(s) affected: `SettleLayer.sol`

Description: When creating a pool, the caller can choose any combination of `ERC20` like contracts. However, those contracts might not be fully `ERC20` compliant, might be malicious, or have implementation bugs. If that's the case, performing operations on them might leave pools with balances different than what CrocSwap expects. In the worst case, this could lead to a loss of funds. **Note:** This is only marked as `Informational` as CrocSwap verifies that balances are correct when the pool is created (`SettleLayer.sol` - `L141-142`). However, this can be insufficient; other transfers can still behave unexpectedly.

Recommendation: Consider creating a list controlled by CrocSwap governance with the tokens that can be used to create pools. Alternatively, consider adding checks to ensure that after performing an operation on a CrocSwap pool, the final token balance matches what CrocSwap expects.

Update: Acknowledged by the developers and added additional documentation (`TokenModel.md`) concerning unsafe tokens.

QSP-24 `collectEther` Should only Be Called Once

Severity: *Informational*

Status: Fixed

Description: The methods that deal with `msg.value` should never be called more than once to avoid double spend (`depositSurplus`, `transactEther`, and `collectEther` - list might be incomplete). Given that the code is quite complex and big, it might be possible that future code refactors allow calling these methods more than once by mistake. This could affect the balances tracked by CrocSwap and have severe consequences.

Recommendation: Consider getting `msg.value` from a single method and ensure that it can only be called once per transaction. Also, add tests to validate that these methods can never be called more than once.

Update: Fixed in commit ed7ba22 by guarding `msg.value` calls behind one access function and reverting on multiple calls.

QSP-25 `unchecked` Keyword Risk

Severity: Informational

Status: Mitigated

File(s) affected: `contracts/*`

Description: The `unchecked` keyword is constantly used across the codebase to reduce gas consumption. However, this also places the responsibility of preventing overflows and underflows on CrocSwap's code. This is a huge risk, as there are too many methods using this keyword, so it's quite possible that there are overflow/underflow bugs hidden throughout the code or that are introduced in future upgrades. This could have severe consequences.

Recommendation: We recommend removing the `unchecked` keyword from most places and instead pay the additional gas. This can help avoid subtle bugs that are hard to detect. If the CrocSwap team decides to take the risk, consider adding tests for each method using this keyword to ensure that it's never possible to over/under flow and that all arithmetic operations are always safe.

Update: Mitigated in commit 4e58efd by removing some instances of `unchecked` code blocks and commenting others, explaining why they may not overflow/underflow.

QSP-26 Risk when Passing Structs

Severity: Informational

Status: Acknowledged

File(s) affected: `contracts/*`

Description: When structs are passed with the `memory` keyword as a parameter to another function, any changes in the struct will only be reflected in the original struct if the function lives within the same contract. If the function called lives in a different contract or in a separate library (and the function in the library is not marked as `internal`), any changes to the struct won't be reflected back.

Recommendation: Given Solidity's rules in this area are so complex, be very careful when passing structs around. Make sure to add unit tests to validate that when structs are passed as a parameter to a function they are updated properly.

Update: Acknowledged by the developers.

QSP-27 Risk Using Constants for Command Codes

Severity: Informational

Status: Fixed

File(s) affected: `contracts/callpaths/*`

Description: The command codes in CrocSwap are mapped to constants to improve readability. However, this could be an issue if there is more than one command code with the same number (the compiler won't ever complain if there are more than two command codes with the same value). This might be harmful depending on which code is duplicated.

Recommendation: Consider using enums instead of plain constants. Otherwise, be extremely careful when assigning constants and constantly review that there are no duplicates.

Update: Fixed in commit 1b17868 by replacing the constant numbers with newly declared constant variables in `WarmPath.sol`.

QSP-28 `deltaQuote` Method Is Called with the Wrong Parameters

Severity: Informational

Status: Fixed

File(s) affected: `CurveMath.sol`

Description: The `deltaQuote` method (`L176`) expects the second parameter to be the price and the third one to be `limitPrice`. However, when `calcLimitFlows` (`L152`) calls `deltaQuote` it's mixing the order of the parameters; it's providing `limitPrice` as the second and `price` as the third.

Recommendation: Update `calcLimitFlows` (`L152`) to make sure it's calling `deltaQuote` in the right way. Also, add unit tests to ensure that methods are calling other functions in the correct way.

Update: Fixed in commit 87518dd by changing the parameter order, as suggested.

QSP-29 `isPoolInit` Will Be Incorrect if New Schemas Are Added

Severity: Informational

Status: Fixed

File(s) affected: `PoolRegistry.sol`

Description: The `isPoolInit` methods (`L300` and `L307`) return true if the pool's schema matches the `BASE_SCHEMA` (value 1), otherwise it returns false. However, this will only work as long as there is only one schema in CrocSwap. As soon as a new schema is added the result will be incorrect.

Recommendation: Consider refactoring these methods (now or when a new schema is added) to return false if the schema is 0, revert if an invalid schema is provided, and true otherwise. Also, add tests to validate that this method works properly across all cases.

Update: Fixed in commit f1ab8b0 by reverting if a schema higher than the base schema is provided.

QSP-30 Use of `call` when Transferring Ether

Severity: *Informational*

Status: Mitigated

File(s) affected: `TransferHelper.sol`

Description: `TransferHelper.safeEtherSend` must be used only with non-reentrant functions that must respect the Checks Effects Interaction pattern since it forwards all the gas to the receiving address. Even if the functions are non-reentrant, and if a Dapp state cannot be tampered with, a global state that includes third-party applications can still be tempered.

Recommendation:

Update: Mitigated by refactoring code to conform to the checks-effects-interaction pattern (See QSP-10).

QSP-31 Missing Overflow Check in `LiquidityMath.shaveRoundLotsUp()`

Severity: *Undetermined*

Status: Fixed

File(s) affected: `LiquidityMath.sol`

Related Issue(s): SWC-101

Description: Function `LiquidityMath.shaveRoundLotsUp()` "shaves" off liquidity smaller than the defined active "lots" bits (10 + 1 bit for knockout liqidity) and rounds up the resulting lot-multiple, by adding one. However, any value for `liq` with a mask of `0xfffffffffffffffffffffffffff800` would lead to an overflow and resulting in returning `0`.
Note that removing the `unchecked {...}` keyword and thereby activating the implicit arithmetic over-/underflow checks of solidity >= 0.8.0 would not help here, as the overflow happens on the intermediately shifted `liq` value.

Recommendation: We recommend checking `liq` to not have said masking bits set, i.e. via `require((liq & 0xfffffffffffffffffffffffffff800) != 0xfffffffffffffffffffffffffff800, "overflow");`.

Update: Fixed in commit f877a04 by a threshold check, as suggested.

QSP-32 Use of Unsafe Cast Operation in `Chaining.pinFlow()`

Severity: *Undetermined*

Status: Fixed

File(s) affected: `Chaining.sol`

Description: Function `Chaining.pinFlow()` casts the parameter `uint128 qty` using `int128(qty)` in L418, `L419`, `L420` and `L421`, without properly checking if it fits within `type(int128).max`, potentially allowing overflows when casting.

Recommendation: Consider replacing the primitive cast operation (`int128(...)`) with i.e. the safe cast library function (`.toInt128()`) from OpenZeppelin or checking whether it fits within `type(int128).max`, while update the unit test accordingly.

Update: Fixed in commit c91fb48 by making use of the safe cast operation `toInt128Sign()`.

QSP-33 Missing Licenses

Severity: *Undetermined*

Status: Fixed

File(s) affected: `contracts/*`

Description: While some contract files are marked to be licensed according to `GPL-2.0-or-later` (`TickMath.sol`, `SafeCast.sol`, ...) as marked using the `// SPDX-License-Identifier` tag on top of the contract, most files are marked as `Unlicensed`.

Recommendation: Consider clarifying this inconsistency before releasing the code to the public and adding a suitable license accordingly.

Update: Fixed in commit 3875bef by changing the licenses to `GPL-3`.

QSP-34 Several Methods Are Unclear

Severity: *Undetermined*

Status: Mitigated

File(s) affected: `SwapCurve.sol`, `CurveAssimilate.sol`, `CurveRoll.sol`, `MarketSequencer.sol`, `LevelBook.sol`, `Chaining.sol`, `CurveMath.sol`, `LiquidityMath`

Description: There are several methods across the codebase whose behavior is not entirely clear. The following list contains some of the methods:

1. `SwapCurve.sol`:`boundLimit`(`L204`) - It's not clear why this line is subtracting one from the sqrt price. **Recommendation:** Consider adding a comment explaining why this is needed.
2. `CurveAssimilate.sol`:`adjustConcRewards`(`L185`) - It's unclear where the formula described on `L190` is derived. **Recommendation:** Consider adding a comment or an external document explaining how this (and other formulas) are derived.
3. `CurveRoll.sol`:`calcBaseFlowPrice`(`L273`) - According to the documentation, this method should favor base token over-collateralization. Intuitively, this should mean that when the price is going up, the method rounds up. (`L282` should be `price + priceDelta + 1`). However, it's unclear why that's not the case. **Recommendation:** Consider adding documentation along with examples that help explain why this method behaves like this.
4. `MarketSequencer.sol`:`applyConcentrated`(`L341`) - It's unclear why `markPosAtomic` is needed when applying concentrated liquidity on long-range, but not when minting concentrated positions on regular orders. **Recommendation:** Consider adding more documentation explaining this.
5. `CurveRoll.sol` - It's not clear why the `burnSwap` value is 0 if the `isBaseQty` is true in `L149` (`setShaveUp`) or when it's false on `L136` (`setShaveDown`). **Recommendation:** Consider documenting the reason why and adding test cases to validate that the code is behaving as expected.

6. `LevelBook.sol` - It's not clear why the `feeOdometer_` is initialized to `feeGlobal` on `L150` and why `L236` allows the `clockFeeOdometer` to underflow. **Recommendation:** Given the fee odometer functionality is so complicated, consider adding more documentation along with a few examples (when `tick` is above `midTick`, when `tick` is below, and when there are overflows).
7. `Chaining.sol:determinePriceRange(L347)` - It's unclear why the `askPrice` (`L358`) and `bidPrice` (`L360`) are set to `curvePrice`. Consider documenting how (and why) the price is chosen depending on the value of the parameters and add examples for each of the cases.
8. `CurveMath.sol:priceToTokenPrecision(L349)` - The formula used to calculate the number of tokens to burn to over-collaterize when the price is below `Q64` is quite confusing (described in `L370-L374`). Moreover, the operations performed on `L377` do not match that formula. Finally, the method can return `type(uint128).max` if `L377` returns a very large value. It's unclear if this could ever happen in practice; if it can, this would be a big problem since the return value is supposed to be quite small. **Recommendation:** Consider reviewing the operations performed to ensure that they are correct. Then update the documentation accordingly (along with examples) and add tests to validate that the method behaves properly across all possible cases (e.g., when the liquidity is very high or the price very low).
9. `LiquidityMath.sol:liquidityToLots(L76)` - This method receives a parameter called `liq`, truncates the last 10 bits in the number, and then validates that the original `liq` matches the truncated value (`L82`). This means that the `liq` given must always be a multiple of 1024; it's unclear if this is by design or if it's a mistake. **Recommendation:** Review if this is the correct behavior. If it is, make sure that all the callers of this method are aware of this requirement and that they deal with it appropriately.

Recommendation: Consider following the recommendations described above.

Update: Mitigated in commits 897b626, 050376f, f8e8ab4 and 3365689 by adding additional inline code comments and external documentation to most of the listed functions.

- `SwapCurve.sol:boundLimit`(L204): Fixed.
- `CurveAssimilate.sol:adjustConcRewards`(L185): Fixed.
- `CurveRoll.sol:calcBaseFlowPrice`(L273): Fixed.
- `MarketSequencer.sol:applyConcentrated`(L341): **Unresolved**.
- `CurveRoll.sol`: Fixed.
- `LevelBook.sol`: **Unresolved**.
- `Chaining.sol:determinePriceRange`(L347): Fixed.
- `CurveMath.sol:priceToTokenPrecision`(L349).
- `LiquidityMath.sol:liquidityToLots`(L76): Fixed.

QSP-35 Liquidity 0 Is Allowed

Severity: *Undetermined*

Status: Fixed

File(s) affected: `CurveAssimilate.sol`, `CurveMath.sol`, `CurveRoll.sol`

Description: There are several places in the code with special cases for when liquidity is 0 (`CurveAssimilate.sol:L47`, `CurveMath.sol:L313`, `CurveRoll.sol:L275`). It's unclear which cases would ever allow a curve to have zero liquidity.

Recommendation: Assess if 0 liquidity is ever valid. If it is, add documentation to explain how it can be ever reached. Otherwise, revert the transaction if any of these methods ever receives zero liquidity if it's never valid.

Update: Fixed in commit 146ab2b by enforcing 1-liquidity when trying to initialize `initCuve()` with zero.

QSP-36 Possible to Call Different Versions of `HotPath`

Severity: *Undetermined*

Status: Fixed

File(s) affected: `HotPath.sol`, `CrocSwapDex.sol`

Description: The `hotPathOpen_` flag is used to determine which version of the `HotPath` is enabled (the one directly embedded on the `CrocSwapDex` contract or the one as a sidecar). However, this flag is only checked on the `CrocSwapDex` contract itself (`L92`). The `userCmd` on the `HotProxy` is not checking this flag. If the flag is set to true and the `HotProxy` is deployed as a sidecar, users could choose which one to use; it's unclear if this could cause any side effects.

Recommendation: Consider adding a check to swap in the `HotProxy` to ensure that it can only be called if `hotPathOpen_` is not enabled.

Update: Fixed in commit 608b03d by adding a check on the flag within `HotPath.sol`, as suggested.

QSP-37 Use of Virtual Tokens Appears Unnecessary

Severity: *Undetermined*

Status: Fixed

File(s) affected: `DepositDesk.sol`

Description: The usage of virtual tokens in CrocSwap is not very clear. CrocSwap only keeps a balance of the virtual tokens, but it does not know anything about the actual tokens. Because of this, everything is dependent on an external tracker contract. If that contract is not well implemented, the balances tracked in CrocSwap can be incorrect. Moreover, the virtual tokens are not used anywhere else, so they appear to be unnecessarily increasing the attack surface.

Recommendation: Assess whether virtual tokens are really needed. If they are not needed, remove the code.

Update: Fixed in commit e3a0b25 by removing virtual token functionality.

QSP-38 Virtual Deposit Can Be Exploited

Severity: *Undetermined*

Status: Fixed

File(s) affected: `ColdPath.sol`

Description: `ColdPath.depositVirtual()` and `ColdPath.disburseVirtual()` allow the user to input the tracker address when calling `userCmd()` and possibly modify the contract state by using a dummy tracker that is not whitelisted. However, the usability of the virtual deposit feature is unclear.

Recommendation: The team should clearly document all possible cases where this issue can represent a risk since we are not aware of the usability of the virtual deposit feature.

Update: Fixed in commit e3a0b25 by removing virtual token functionality (See QSP-40 "Use of Virtual Tokens Appears Unnecessary").

QSP-39 Incorrect Condition in `TradeMatcher.sweepSwapLiq()`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `TradeMatcher.sol`

Description: The implemented `sweepSwapLiq()` function contains the following requirement:

```
require(swap.isBuy_ ? curve.priceRoot_ < swap.limitPrice_ :
+                             curve.priceRoot_ > swap.limitPrice_, "SD");
```

However, it does not cover the case when both values are equal, which can happen following the user inputs.

Recommendation: Include the equality to avoid the transactions failing due to an uncovered case.

Update: Fixed in commit edbab02.

QSP-40 Unclear Usage of the Fallback Function

Severity: *Informational*

Status: Fixed

File(s) affected: `TimeLock.sol`

Description: `fallback()` function was added along `receive()` to the cloned `TimeLock` contract. `receive()` can be removed in this case since `fallback()` can execute any external call that is not intended to an implemented function in the `TimeLock` contract including empty calldata calls.

Update: Fixed in commit 7621147.

QSP-41 Documentation Issues and Best Practices (Fix Review)

Severity: *Informational*

Status: Fixed

Description:

1. Incorrect comments added (copy-and-paste) in `Encoding.sol#L82`, `L99`, `L114`, `L120` and `L133` stating `Incrementing by 32 at a time should never overflow 256 bits` - However, they add multiples of `32` (`32*x`) and not `32`.
2. NatSpec parameter comment in `AgentMask.sol#L49` not changed from `salt` to `callPath`.
3. NatSpec comment in `AgentMask.sol#L356` not changed from `salt` to `call path`.
4. The added NatSpec comment in `AgentMask.sol#L19` is missing the word `liquidity` (`tokens and owned` -> `tokens, and liquidity owned`).
5. `LongPath.userCmd()` is now missing a NatSpec comment for the new return value `int128[] flows`.
6. The fix for `ColdPath.sol:setNewPoolLiq` introduces a new state variable `uint constant MAX_INIT_POOL_LIQ = 10_000_000` in contract `PoolRegistry.sol` without explicit variable width.
7. Fix introduced a typo in `SwapCurve.sol#L197`: `asymettry` -> `asymmetry`.
8. Fix introduced a typo in `Chaining.sol#L351`: `requires` -> `required`.
9. Code still contains references to the virtual token:
    - `DepositDesk.sol#L7`, `L80`, `L104`.
    - `ICrocVirtualToken.sol`.
10. `PoolRegistery.applyDiscount()` uses a description for discount that is not mentioned in `ICrocPermitOracle(pool.oracle_).checkApprovedForCrocPool()` where it says that the function should throw by itself.
11. `callCath` is used as a salt in `AgentMask.reEntrantApproved()` but the naming in the NatSpecs was not changed.
12. Unused `LiquidityDirective` structure in `Direcrives.sol`, still used in the test files.

Update: Fixed in commit d397fdd.

Automated Analyses

Slither

Slither did not raise any significant findings.

Adherence to Specification

1. Long form orders are passed as an array of bytes encoded according to the set of rules defined in `Encoding.md` and the documentation online. However, there are several inconsistencies:

   . The documentation wrongly states that the dust threshold in the settlement directive is of type `int128`. However, in the code, it is a `uint128`.

   . The documentation marks the `poolIdx` in the `PoolDirective` as a `uint24` (3 bytes). However, the code uses `uint256` (32 bytes).

   . In the documentation online, the `ConcentratedDirective` struct is called Range Bookend Directive, and the parameters described are different from the ones found in the code. **Note:** The `ConcentratedDirective` struct defines the `isTickRel` and the `isAdd` flags in a different order than how they are parsed. While this is not an issue in itself, it might highlight that there's a different expectation on the order of the fields.

   . The documentation online claims that the `SwapDirective` has an unused field of 8 bytes called `mask`. However, the code is not parsing or skipping this field. It's unclear if this is missing or the documentation online is just outdated.

   . The Settlement Directive is called `SettlementChannel` in code. It's unclear which one is correct.

   Recommendation: Review the `Encoding.md` and the documentation online to ensure they are in alignment and that the code is decoding the order accordingly.

2. `AgentMask.sol` - According to the comment on `L254-253` the address that will receive the tip for relaying a message can have two "magic values": 0x10 (will transfer the tip to `msg.sender`) and 0x20 (will transfer the tip to `tx.origin`). However, the `markTipRecv` is using different values; it's using `MAGIC_SENDER_TIP` (equals to 0x100) and `MAGIC_ORIGIN_TIP` (equals to 0x200). **Recommendation:** Determine which value is correct and adjust code and documentation accordingly (consider updating external documentation too).

3. `ColdPath.sol` - The documentation for the `approveRouter` (`L337`) is incorrect. It describes the `forDebit` and `forBurn` parameters. However, those parameters do not exist. Moreover, the method accepts a `salt` parameter that is supposed to help control the permissions given to a particular router (to perform a specific type of operation). However, this functionality is not implemented anywhere. As is, the `salt` parameter is pretty much useless. **Recommendation:** Consider reviewing if limiting the router permissions is needed or not. If it is, implement it. Otherwise, update the documentation accordingly.

4. `ICrocLpConduit.sol` - According to the interface documentation, when calling `depositCrocLiq` on an ambient position, the milage parameter should be zero. However, when calling this method, the `TradeMatcher.sol` is passing the `deflator` to this parameter (`L184` and `L193`). Therefore, it's not going to be 0. It's unclear what the consequences are for this mismatch. The same happens with `withdrawConduit`. **Recommendation:** Consider what is the right value to pass to the `milage` parameter for ambient positions. Then update code and documentation accordingly (and add unit tests).

Code Documentation

1. Missing or incorrect NatSpec comments:
   1. `Bitmaps.isBitSet()`: Missing NatSpec comment for return value.
   2. `Bitmaps.isTickFinite()`: Missing NatSpec comment for parameter `tick`.
   3. `Directives.ConcentratedDirective`: Missing NatSpec comment for struct member `isTickRel_`.
   4. `Directives.AmbientDirective`: NatSpec comment for struct member `liquidity_` states `If zero, this is a non-action`. However, zero, together with `isAdd_` being false, indicates `the use of a flexible back-filled rolling quantity in place`, as mentioned on L74/L75.
   5. `TransferHelper.safeTransferFrom()` contains copy-and-pasted and unadjusted NatSpec comments from `safeTransfer()`.
   6. `KnockoutLiq.KnockoutPos`: NatSpec name for structure element `timestamp_` has a typo (`timetamp_`).
   7. `KnockoutLiq.encodePivotKey()`: Missing NatSpec comment for return value.
   8. `KnockoutLiq.encodePosKey()`: Incorrect order of NatSpec parameters (`pivotTime` is last, not second parameter) and missing NatSpec comment for return value.
   9. `KnockoutLiq.assertValidPos()`: Explained structure for `knockoutBits` does not match parsing in `KnockoutLiq.unpackBits()`.
   10. `KnockoutLiq.unpackBits()`: Missing NatSpec parameter comment for parameter `knockoutBits` and last return value `onGrid`.
   11. `CompoundMath.approxSqrtCompound()`: Parameter comment for `x64` wrongly listed as `x` and return value comment states it returns a `128-bit fixed point`. However, it only returns a 64-bit value.
   12. `CompoundMath.compoundShrink()`: Incorrect parameter names for `val` and `deflator`.
   13. `CurveMath.CurveState`: Inconsistent struct member names in NatSpec comments for `ambientSeeds_` and `concLiq_`.
   14. `CurveMath.liquiditySupported()`: Parameter `priceX` and its explanation are listed twice, instead of `priceY`.
   15. `PriceGrid.isOnGrid()`: Parameter names for `lowerTick` and `upperTick` are incorrect (copy-and-paste from `PriceGrid.verifyFit`).
   16. `PoolSpecs.Pool`: Missing NatSpec comment for struct member `knockoutBits_`.
   17. `Chaining.plugLiquidity()` (L173): Missing NatSpec comment for parameter `dir`.
   18. `Chaining.plugLiquidity()` (L209): Missing NatSpec comment for parameter `bend` and incorrect order for parameter `flow`.
   19. `Chaining.sizeAmbientLiq()`: Typo for parameter name `inBaseQty` (`isBaseQty`).
   20. `Chaining.sizeConcLiq()`: Typo for parameter name `inBaseQty` (`isBaseQty`).
   21. `Chaining.scalePlug()`: Typo for parameter name `rollGap` (`roll`).
   22. `KnockoutFlagPath.crossCurveFlag()`: Missing NatSpec comment for return value.
   23. `KnockoutLiqPath.claimCmd()`: Missing NatSpec comment for return value.
   24. `KnockoutLiqPath.recoverCmd()`: Missing NatSpec comment for return value.
   25. `ColdPath.setTemplate()`: Spurious NatSpec parameter comments for `protocolTake` and `permitOracle`.
   26. `ColdPath.revisePool()`: Spurious NatSpec parameter comment for `protocolTake` and missing parameter comment for `knockout`.
   27. `ColdPath.depositSurplus()`: Spurious NatSpec parameter comment for `isTransfer`.
   28. `ColdPath.approveRouter()`: Mixup of NatSpec `param` and `notice` keywords and/or incorrect parameters `forDebit` and `forBurn`.
   29. `CrocSwapDex.protocolCmd()`: Missing NatSpec comment for return value.
   30. `PoolRegistry.setPoolTemplate()`: Missing NatSpec comment for parameters `knockout`, `oracleFlags` and spurious parameter comment `permitOracle`.
   31. `PoolRegistry.setPoolSpecs()`: Missing NatSpec comment for parameters `poolIdx`, `knockoutBits` and spurious parameter comment `protocolTake`.
   32. `PoolRegistry.queryPriceImprove()`: Missing NatSpec comment for return value.
   33. `PoolRegistry.queryPool()`: Missing NatSpec comment for return value.
   34. `PoolRegistry.selectPool()`: Missing NatSpec comment for return value.
   35. `PositionRegistrar.harvestPosLiq()`: Missing NatSpec comment for parameter `feeMileage`.

36. `PositionRegistrar.changePosOwner()`: Incorrect order for parameter `poolIdx`.
37. `HotPath.pivotOutFlow()`: Missing NatSpec comments for parameters `flow`, `minOutput`, `isBuy` and `inBaseQty`.
38. `MicroPaths.burnRange()` and `MicroPaths.mintRange()`: Missing NatSpec comment for parameter `priceTick`.
39. `DepositDesk.depositSurplusPermit()`: Incorrect/Incomplete NatSpec parameter formatting.
40. `DepositDesk.disburseSurplus()`: Spurious NatSpec parameter comment `owner`.
41. `LiquidityCurve.liquidityReceivable()` (L104): Missing NatSpec comment for return value.
42. `CrocPolicy.CrocResolutionTreasury()`: Missing NatSpec comment for event parameter `sudo`.
43. `CrocPolicy.CrocPolicySet()`: Missing NatSpec comment for event parameter `proxyPath`.
44. `CrocPolicy.CrocPolicyForce()`: Missing NatSpec comment for event parameter `proxyPath`.
45. `CrocPolicy.constructor()`: Missing NatSpec comment for parameter `dex`.
46. `CrocPolicy.opsResolution()`: Missing NatSpec comment for parameter `proxyPath`.
47. `CrocPolicy.treasuryResolution()`: Missing NatSpec comments for parameters `proxyPath` and `sudo`.
48. `CrocPolicy.invokePolicy()`: Missing NatSpec comment for parameter `proxyPath`.
49. `CrocPolicy.setPolicy()`: Missing NatSpec comment for parameter `proxyPath`.
50. `CrocPolicy.forcePolicy()`: Missing NatSpec comment for parameter `proxyPath`.
51. `CrocPolicy.emergencyReset()`: Missing NatSpec comment for parameter `proxyPath` and incorrrect entry for parameter `reason` (`policy`).
52. `LongPath.userCmd()`: NatSpec keyword mixup `input` -> `param`.
53. `TradeMatcher.mintAmbient()`: Typo for NatSpec parameter comment `lpOwner` (`lpConduit`).
54. `ICrocMinion.protocolCmd()`: Missing NatSpec comment for return value.
55. `ICrocPermitOracle.checkApprovedForCrocPool()`: Missing NatSpec comment for parameter `base`.

2. The following typographical errors have been noted:
    1. L74 of `Directives.sol`: `flxeible` -> `flexible`.
    2. L103 of `Directives.sol`: `swapDeer_` -> `swapDefer_`.
    3. L135 of `Directives.sol`: `ot` -> `of`.
    4. L169 of `Directives.sol`: `hot` -> `hop`.
    5. L12 of `FixedPoint.sol`: `Multiplex` -> `Multiplies`.
    6. L26 of `FixedPoint.sol`: `Dives` -> `Divides`.
    7. L129, L140, L152 and L165 of `Encoding.sol`: `encouded` -> `encoded`.
    8. L189 of `Encoding.sol`: `three` -> `three bytes`.
    9. L18 of `TickMath.sol`: `2^96` -> `2^64`.
    10. L98 and L106 of `LiquidityMath.sol`: `Trunacates` -> `Truncates`.
    11. L115 of `LiquidityMath.sol`: `Gives` -> `Given`.
    12. L123 of `LiquidityMath.sol`: `detla` -> `delta`.
    13. L169 of `LiquidityMath.sol`: `origina` -> `original`.
    14. L176 of `LiquidityMath.sol`: `beggining` -> `beginning`.
    15. L16 of `ProtocolCmd.sol`: `transfering` -> `transferring`.
    16. L6 of `KnockoutLiq.sol`: `track to` -> `to track`.
    17. L117 of `KnockoutLiq.sol`: `claimint` -> `claimant`.
    18. L150 of `KnockoutLiq.sol`: `hashes hashes` -> `hashes`.
    19. L200 of `KnockoutLiq.sol`: `pivot` -> `pivot time`.
    20. L230 of `KnockoutLiq.sol`: `reverty` -> `revert.`.
    21. L233 of `KnockoutLiq.sol`: `curv'es` -> `curve's`.
    22. L260 of `KnockoutLiq.sol`: `candidates` -> `candidate`.
    23. L23 of `CompoundMath.sol`: `unreasonble` -> `unreasonable`.
    24. L40 of `CompoundMath.sol`: `teh` -> `the`.
    25. L102 of `CompoundMath.sol`: `an` -> `a`.
    26. L111 of `CompoundMath.sol`: `founded` -> `rounded`.
    27. L153 of `CompoundMath.sol`: `liq/* (1 + growth)` -> `liq/(1 + growth)`.
    28. L175 of `CurveMath.sol`: `true true` -> `true`.
    29. L192 of `CurveMath.sol`: `function'` -> `function's`.
    30. L261 and L289 of `CurveMath.sol`: `Calculated'` -> `Calculates`.
    31. L100 of `PriceGrid.sol`: `Calculated'` -> `Calculates`.
    32. L324 of `CurveRoll.sol`: `signFixed` -> `signFlow`.
    33. L336 of `CurveRoll.sol`: `directiona` -> `direction`.
    34. L22 of `CurveCache.sol`: `than` -> `then`.
    35. L23 of `CurveCache.sol`: `prick` -> `price`.
    36. L32 of `CurveCache.sol`: `Give` -> `Given`.
    37. L42 of `CurveCache.sol`: `thereore` -> `therefore`.
    38. L13 of `PoolSpecs.sol`: `markts` -> `markets`.

39. L16 of `PoolSpecs.sol`: `Becuase` -> `Because`.
40. L17 of `PoolSpecs.sol`: `unitializez` -> `uninitialized`.
41. L22 of `PoolSpecs.sol`: `hundredeths` -> `hundredths`.
42. L38 of `PoolSpecs.sol`: `revent` -> `revert`.
43. L113 of `PoolSpecs.sol`: Spurious `is`.
44. L87 of `PoolRegistry.sol`: `burn` -> `initialization`.
45. L141 of `PoolRegistry.sol`: `to much` -> `too much`.
46. L14 of `Chaining.sol`: `convetions` -> `conventions`.
47. L36 of `Chaining.sol`: `denoinate` -> `denominate`.
48. L95 of `Chaining.sol`: Spurious `transaction`.
49. L190 of `Chaining.sol`: `concenrated` -> `concentrated`.
50. L276 of `Chaining.sol`: `menaingless` -> `meaningless`.
51. L296 of `Chaining.sol`: `with` -> `the`.
52. L333 of `Chaining.sol`: `Calculated` -> `Calculates`.
53. L346 of `Chaining.sol`: `inr ange` -> `in range` and `this` -> `the`.
54. L376 of `Chaining.sol`: `beggining` -> `beginning`.
55. L412 of `Chaining.sol`: Spurious `then`.
56. L10 of `TokenFlow.sol`: `length` -> `long`.
57. L31 of `TokenFlow.sol`: `tthat` -> `that`.
58. L32 of `TokenFlow.sol`: `bookeeping` -> `bookkeeping`.
59. L47 of `SwapCurve.sol`: `a the` -> `the`.
60. L78 of `SwapCurve.sol`: `forthis` -> `for this`.
61. L107 of `SwapCurve.sol`: `imapct` -> `impact`.
62. L109 of `SwapCurve.sol`: `udpate` -> `update`.
63. L121 of `SwapCurve.sol`: `posistive` -> `positive`.
64. L172 of `SwapCurve.sol`: `reacking` -> `reaching`.
65. L269 of `SwapCurve.sol`: `neveer` -> `never`.
66. L79 and L149 of `MicroPaths.sol`: `burn` -> `mint`.
67. L17 of `DepositDesk.sol`: `lateral` -> `later`.
68. L21 of `DepositDesk.sol`: `owner` -> `recv`.
69. L146 of `DepositDesk.sol`: `value` -> `size`.
70. L163 of `DepositDesk.sol`: `trasnfer` -> `transfer`.
71. L26 of `SafeCast.sol`: `uint182` -> `uint192`.
72. L95 and L122 of `LevelBook.sol`: `liq` -> `lots`.
73. L218 of `LevelBook.sol`: `lowerTick` -> `upperTick`.
74. L24-L29 of `PositionRegistrar.sol`: Incorrect numbering.
75. L168 of `PositionRegistrar.sol`: `speciic` -> `specific`.
76. L30 of `StorageLayout.sol`: `than` -> `then`.
77. L41 of `StorageLayout.sol`: `transfered` -> `transferred`.
78. L157 of `StorageLayout.sol`: `upgradears` -> `upgraders`.
79. L156 of `KnockoutPath.sol`: `undrlying` -> `underlying`.
80. L167 of `KnockoutPath.sol`: `becuase` -> `because`.
81. L115 of `ColdPath.sol`: `initialiaze` -> `initialize`.
82. L131 of `ColdPath.sol`: `insatiated` -> `instantiated`.
83. L40 of `CrocSwapDex.sol`: Spurious `will`.
84. L75 of `CrocSwapDex.sol`: `revery` -> `revert`.
85. L181 of `CrocSwapDex.sol`: `contrurctor` -> `constructor`.
86. L12 of `CrocPolicy.sol`: `an` -> `can`.
87. L161 of `CrocPolicy.sol`: `of` -> `or`.
88. L167 of `CrocPolicy.sol`: `commited` -> `committed`.
89. L168 of `CrocPolicy.sol`: `mandata` -> `mandate`.
90. L201 of `CrocPolicy.sol`: `authroity` -> `authority`.
91. L18 of `ICrocLpConduit.sol`: `the position` -> `position`.
92. L22 of `ICrocLpConduit.sol`: `lower` -> `upper`.
93. L19 of `ICrocMinion.sol`: `transers` -> `transfers`.
94. L25 and L52 of `ICrocPermitOracle.sol`: `disriminate` -> `discriminate`.
95. L11 of `ICrocVirtualToken.sol`: `token token` -> `token`.
96. L38 of `ProtocolAccount.sol`: `receipient` -> `recv`.

3. The `@dev` comment on L19 of `TickMath.sol` states `Throws if |tick| > max tick`. However, this check was removed from the original Uniswap file and instead

replaced with a check `tick >= MIN_TICK && tick <= MAX_TICK` (as they are no longer equal in absolute size). Consider adjusting the comment accordingly.

4. Comment on L60 of `LiquidityMath.sol` states that max lot of `2^96` is `equivalent to 2^108 of liquidity`. However, given one lot is equal to 1024 of liquidity, `2^96` lot equals `2^106` liquidity, not `2^108`. Consider adjusting the comment.
5. Comment on L22 of `ProtocolCmd.sol` is a copy-and-paste of L20 and does not reflect the corresponding line of code defining `SAFE_MODE_CODE`.
6. Comment on L177 of `KnockoutLiq.sol` states that `KnockoutLiq.encodeChainLink()` tightly packs the pivot time and the fee mileage, but it also packs in the given salt.
7. The documentation for knockout liquidity and long path (long form orders) is quite limited. Given these are complicated topics (building on top of other complex topics) it might be quite helpful to expand your technical docs with documents about them.
8. `KnockoutLiq.sol` - The documentation of the `KnockoutPosLoc` (`L73`) is describing the parameter `rangeTicks_`, which does not exist in the struct (instead, there is `upperTick_`). It's unclear if this is an outdated comment or if there's something missing in the code. Determine which parameter is correct and update the documentation/code accordingly.
9. `KnockoutLiq.sol` - The documentation for the `knockoutBits` parameter on `L238` is incorrect. The document claims that the last two bits are unused. However, that's not the case. The `unpackBits` is using the 6th bit (7th if you start counting bits from 1) to determine the value of the `onGrid` flag.
10. `Bitmaps.sol` - The bitmaps functionality is quite complex to understand. The CrocSwap documentation could benefit from additional information. A running example showing how a particular tick is mapped into a bitmap and how the system would behave to find the next bitmap (for both buys and sells) could be quite useful. Also, consider documenting how the system behaves when there are spills over boundaries (over terminus, mezzanine, and lobby).
11. `SwapCurve.sol` - The comment for `swapToLimit` on `L39` states that the caller should check the `qtyLeft_` field. However, the `Chaining.PairFlow` struct does not contain this field. It's unclear if this is due to some refactoring and the comment was never updated or if it's due to something missing in the code. Consider reviewing this to ensure that the code is correct and update the documentation accordingly.
12. `CurveMath.sol` - The comment on `L224` wrongly states that the result is guaranteed to be 196 bits. That is wrong; it's guaranteed to be 192 bits.
13. `WarmPath.sol` - Missing documentation for parameter `lpConduit` on `L264` and `L153`.
14. `LevelBook.sol` - The comment on `L95` describes a parameter called `liq` that does not match the actual parameter used (`lots`).
15. `KnockoutLiq.sol` - Consider documenting the parameters of the `spreadOkay` method (`L276`). In particular, it's unclear what the `yonder` parameter represents.
16. `Directives.sol` - The comment on `L67-L68` is not related to any struct or other piece of code. It appears to be wrongly left after some code refactoring. Consider removing it if it's not needed.
17. `TickCensus.sol` - The documentation for the `doesSpillBit` method (`L156`) and the `isSpill` parameter on `L126` are unclear. Consider reviewing them.
18. `SettleLayer.sol` - `L238` mentions that this is the only place in a standard transaction where the `msg.value` is read. However, this value is also read on the `depositSurplus` method on `DepositDesk.sol`.
19. `CrocSwapDex.sol` - The comment on `L112` appears to be incorrect. It's unclear what the 64 spill sidecars are and why they are mentioned as unused.
20. `StorageLayout.sol` - According to the comment on `L30`, the default value for `hotPathOpen_` is false, and when this variable is set to true the embedded hot-path is not enabled; both of these statements are wrong.
21. `CrocEvents.sol` - Consider documenting the events in this file so that it's clear what they represent.
22. `PoolRegistry.sol` - The documentation on the `verifyPermitInit` method appears incorrect. This method is called when initializing the pool, not when burning liquidity.
23. `LiquidityCurve.sol` - The documentation for the `initPrice` (`L288`) states that the `priceRoot` is represented as a 96-bit fixed point. However, that's not correct; It's represented as a `Q64.64` (128-bit).
24. `SettleLayer.sol` - The documentation for the `settleInitFlow` method on `L133` states that the `baseFlow` can be negative for credits paid to user. However, this method is used to settle the initial flow. Therefore, it's never possible to pay credits to the user; the flow does not have any funds to pay.
25. `ColdPath.sol` - Documentation for the `depositSurplus` method (`L268`) appears to be incorrect. It mentions that this method is used to pay out or pay in surplus collateral directly. However, this method is only used to pay in surplus collateral. Also, the documentation for the parameters is wrong.
26. . `DepositDesk.sol` - The documentation for the `transferSurplus` method on `L68` wrongly describes the `to` param as the user account from which the surplus collateral will be sent. Instead, it's the account that will receive the surplus collateral.
27. `Chaining.sol` - `L346`. It's unclear what the comment "price is sitting in range or not" is supposed to say. There are several typos that make it hard to understand.
28. `ProtocolCmd.sol` - The comment at `L9` appears incomplete. It says "... the top-level interface to the Croc dex contract contains a general purpose."
29. `ICrocMinion.sol` - `L15` and `L17` missing closing parentheses.
30. We identified several typos across the codebase:
    - `Layout.md`
      - L29 - Typo "exeucting" should be "executing".
    - `MarketSequencer.sol`
      - L21 - "responsibile" should be "responsible" and "sequetial" should be "sequential".
    - `ColdInjector.sol`
      - L14 "involed" should be "invoked"
    - `AgentMask.sol`
      - L76: "who's" should be "whose".
      - L37: "who's" should be "whose".
    - `WarmPath.sol`
      - L147: "par" should be "pair".
    - `ColdPath.sol`
      - L132: "continue exist" should be "continue to exist".
      - L336: "belongining" should be "belonging".
    - `CrocSwapDex.sol`
      - L102: "sidecare" should be "sidecar".
    - `PoolSpecs.sol`
      - L11 "Specifcations" should be "Specifications"

.
    - L59: “about on a” should be “about an”.

. `PoolRegistry.sol`
    - L106: “beore” should be “before”.
    - L165: “permanetely” should be “permanently”.
    - L204: “storage” should be “store”.
    - L188: “arbitararily” should be arbitrarly.
    - L305: “initailized” should be “initialized”.

. `CrocPolicy.sol`
    - L20: “@notie” should be “@notice”.

. `ProtocolAccount.sol`

. `MarketSequencer.sol`
    - L45: “sequences of action” should be “sequence of actions”.
    - L48: “gap-failled” should be “gap-filled”.
    - L65, L118, L174, L202, L229: “speciication” should be “specification”.
    - L232: “permanetely” should be “permanently”.
    - L233: “Represeted” should be “represented”.
    - L252: "Appplies” should be “Applies”.

. `TradeMatcher.sol`
    - L76: “permanetely” should be “permanently”.
    - L88: “responsbility” should be “responsibility”.
    - L114: “prickTick” should be “priceTick”.
    - L146: “off of” should be “of”.
    - L150: “responsbility” should be “responsibility”.
    - L160: “operations” should be “operation”.
    - L127: “thhis” should be “this”.
    - L315: “ultaimtely” should be “ultimately”.
    - L371: “we calculating” should be “calculating”.

. `FixedPoint.sol`

. `SettleLayer.sol`
    - L126: “associated with a the initailization” should be “associated with the initialization”.
    - L226: “Becaue” should be “Because”.
    - L391: “othersize” should be “otherwise”.

. `DepositDesk.sol`
    - L35: “approviing Permit of the token underlying token ...” should be “approving permit of the underlying token...”.

. `ICrocVirtualToken.sol`
    - L11: The word token is duplicated.

. `ICrocPermitOracle.sol`
    - L25, L52: “disriminate” should be “discriminate”.

. `Directives.sol`
    - L18: “flxeible” should be “flexible”.
    - L31: “exeucting” should be “executing.

. `LiquidityCurve.sol`
    - L31: “ergonomics” should be “economics”.
    - L31: “iterarively” should be “iteratively”.

. `TickMath.sol`
    - L15: “precicion” should be “precision”.

. `TickCensus.sol`
    - L39: “represnted” should be “represented”.
    - L140-L141: Duplicates the word “whether”. Because of this the comment is not clear.

. `Bitmaps.sol`
    - L33: “lefft” should be “left”.
    - L100: “mezznine” should be “mezzanine”.

. `SwapCurve.sol`
    - L269: “neveer” should be “never”.

. `CurveMath.sol`
    - L130: “denomianted” should be “denominated”.
    - L248: “represnted” should be “represented”.
    - L199: “inifnite” should be “infinite”.

.
    - L336-L337: “token” is repeated at the end of L336 and the beginning of L337.
    - L362: “Proivde” should be “Provide” and “roudning” should be “rounding”.

. `CurveAssimilate.sol`
    - L38: “colleted” should be “collected”.
    - L73: “liqudity” should be “liquidity”.
    - L180: “prevous” should be “previous”.
    - L181: “realtive” should be “relative”.

. `CompoundMath.sol`
    - L31: “Tayler” should be “Taylor”.
    - L80: “calulated” should be “calculated”.
    - L155: “allways” should be “always”.

. `CurveRoll.sol`
    - L329: “bothsides” should be “both sides”.
    - L200: “cosntant” should be “constant”.
    - L209: “calulcates” should be “calculates”.
    - L241: “Roudning” should be “Rounding”.
    - L248: “adaquately” should be “adequately”.
    - L249: “roudning” should be “rounding”.

. `LiquidityMath.sol`
    - L52: “liquiidty” should be “liquidity”.
    - L88: “aggergate” should be “aggregate”.
    - L93: “renormalzing” should be “renormalizing”.
    - L148: “postion” should be “position”.

. `PositionRegistrar.sol`
    - L80: “checkpoined” should be “checkpointed”.
    - L197: “necessry” should be “necessary”.

. `LevelBook.sol`
    - L100: “we” should be “the”.
    - L217: “prick” should be “price”.

. `KnockoutCounter.sol`
    - L15: “creditiing” should be “crediting”.
    - L28: “reconstructo” should be “reconstruct”.
    - L320: “perorm” should be “perform”.

. `KnockoutLiq.sol`
    - L235: “@param loc” should be “@param knockoutBits”.

. `KnockoutPath.sol`

. `LongPath.sol`
    - L33: “constitutiin” should be “constituting of”.
    - L78: It’s unclear what “hitherto” refers to.

. `Encoding.sol`
    - L129: "encouded” should be “encoded”.

. `TokenFlow.sol`
    - L66: “Aftering” should be “After”.

. `CurveCache.sol`
    - L42: “conisdered” should be “considered”.

Adherence to Best Practices

1. `MarketSequencer.sol` - The function called `applyConcentrateds` (`L296`) is probably misnamed. It should be `applyConcentrated`.
2. There are several places in the codebase that use "magic" numbers. This is a bad practice, as it makes the code harder to read and understand. Consider changing these numbers for meaningful constants. The following list contains the examples that we have found (list might not be exhaustive):

    . `PoolSpecs.sol`
        - `L117`: This line is doing a bitwise `&` operation against the `0x1` literal.

    . `WarmPath.sol`
        - `L72:L113`

    . `MultiPath.sol`
        - `L30:L54`

    . `SettleLayer.sol`

- L166 and L170

3. There are several functions that share the same name but perform slightly different tasks. For example, `queryPool` in the `PoolRegistry.sol` reverts if the pool is not initialized, while `queryPool` in `PoolSpecs` does not. Be extremely careful when choosing which function to call. Alternatively, consider adopting a clear naming convention to avoid any confusion. Moreover, make sure to add unit tests verifying that the right function is being called.
4. `HotPath.sol` - The `swapDir` method (`L66`) is not specifying the `rollType` to use in the `SwapDirective`. The default value is `NO_ROLL_TYPE` (0 value); it's unclear whether this is by design or an oversight. Therefore, consider setting the value explicitly in the code and add a comment explaining why this value is chosen.
5. `StorageLayout.sol` - The `UserBalance` struct (`L121`) contains separate unrelated fields in a single struct (and a single collection `userBals_`). While this is done for gas savings, using the same collection for separate purposes can easily introduce unexpected bugs. If possible, consider splitting this into separate collections, each having a clear responsibility.
6. It's unclear what is the naming convention for error messages in `require` statements. They are sometimes duplicated, which can make tracing an error more complicated than it should be. Consider documenting how this naming convention works; if it's not good enough, consider adopting a better one.
7. The contracts' names should match the file's name. For example, `ColdInjector.sol` should be called `ColdPathInjector.sol`.
8. The CrocSwap codebase implements a lot of code that is already available in external libraries, like OpenZeppelin. If possible, it can be helpful to take advantage of these libraries to take advantage of the good design/best practices that these libraries offer.
9. The name `ColdPathInjector` is confusing. It's actually a contract that is available to all paths, so it's better to rename it to something more accurate.
10. `SafeCast.sol` - The `timeUint32` is used to convert a timestamp to `uint32` for gas savings. Because of this, this method will work accurately with timestamps smaller than or equal to `type(uint32).max` (this includes all timestamps before the year 2038). After that, the method won't return the right timestamp. **Recommendation:** Assess if the gas savings are significant enough to risk having incorrect timestamps in the future (year 2038). Otherwise, consider increasing it to `uint40` to alleviate the problem in the future.
11. `CrocPolicy.sol` - It's unclear why the `cmdFlags_` under the `PolicyRule` struct is of type `bytes24`, so there can only be 192 possible codes (the commands in the rest of the codebase are `uint8`, with 256 possible values). **Recommendation:** Consider using `bytes32` as that would match the 256 possible values with `uint8`. Otherwise, explain in your developer documentation that commands must be indexed up to 192; anything beyond that would not be possible to be called via `CrocPolicy`.
12. `CrocPolicy.sol` - After an `emergencyHalt` has been dealt with, there's no specific method to return the system to normal. This means that the `treasuryAuth` will have to "manually" disable the safe mode and return the `hotPathOpen_` flag to its previous value.
13. Consider using consistent naming across the codebase. There are different places in the code that use different names for the same variable or concept. For example, the `SettleLayer.sol` uses `recv` and `creditor` in different ways in the same call path. Similarly, ticks are sometimes called `lowTick` or `bidTick`, `highTick`, `upperTick` or `askTick`. This makes the code harder to understand.
14. Consider using `_` to separate large numbers. For example, in `SwapCurve.sol:L266`, the `FEE_BP_MULT` can be written as `1_000_000` (this variable can also be turned into a constant).
15. `WarmPath.sol` - In order to avoid confusions, consider renaming the `mint` function on `L134` to `mintConcentratedLiquidity` while `mint` on `L214` to `mintAmbientLiquidity`. The same can be done with the `burn` method; `burn` on `L160` should be `burnConcentratedLiquidity` and `burn` on `L269` should be `burnAmbientLiquidity`.
16. `TickCensus.sol` - The `seekLobbyDown` (`L269`) and `seekLobbyUp` (`L255`) are using under and overflow as part of their logic. While this might be done for gas reasons, it's not the best practice. Consider testing these methods thoroughly to ensure there are no bugs lurking there.
17. It might be convenient for events to have the `lockHolder_`. That way, we can understand who executed the event.
18. `AgentMask.sol` - The hashes of the `CrocRelayerCall` on `L126`, the `EIP712Domain` on `L137`, and `CrocSwap` on `L139` can be turned to constants. This could help reduce gas consumption when calling commands through a relayer.
19. `AgentMask.sol` - Consider adding the `version` parameter to the `EIP712` signature (`L138`). This can be helpful if a signature for a previous version of the code is not compatible with newer versions of the code.
20. `DepositDesk.sol` - The `applyTransactVal` method has an unusual syntactic sugar (a positive `qty` returns that balance, 0 the full balance, and negative the amount above the value). All methods that use this method need to document this behavior properly. Otherwise, it might be confusing (and dangerous) for the caller of the methods that a value of 0 returns the full balance.
21. `ColdPath.sol` - The parameter `recv` in `L301` is not the receiver of the virtual token. Instead, it is the tracker (the address that will receive the tokens is the `lockHolder_`). Consider renaming the `recv` variable to reflect its purpose properly.
22. Throughout the codebase, several methods with short names are hard to understand. This affects code readability and maintainability. Make sure to use meaningful names for all methods, variables, and constants. The following list contains some of the names that are confusing (the list might not be exhaustive):
    - `SwapCurve.sol`
      - `vigOverSwap`
      - `vigOverFlow`
    - `TradeMatcher.sol`
      - `mintPosLiq`
    - `AgentMask.sol`
      - `casAgent`
      - `casNonce`
23. `PoolRegistry.verifyPermitSwap()` gives a discount but it is not described in the function NatSpecs.
24. `Directives.SwapDirective.inBaseQty_` naming can lead to confusion.
25. All require statements in `ColdInjector.sol` and `TickMath.sol` do not return a full revert message that describe the revert reason.
26. For improved readability and code quality it is advised to remove duplicate or unused code and to not mix test code with production code. In this regard consider the following cases:
    1. Importing `hardhat/console.sol` in `CrocLpErc20.sol`, `CrocSwapDex.sol`, `CrocQuery.sol`, `StorageLayout.sol`, `TickCensus.sol`, `LiquidityCurve.sol`, `ColdInjector.sol`, `TradeMatcher.sol`, `PoolRegistry.sol`, `PositionRegistrar.sol`, `MultiPath.sol`, `MicroPaths.sol`, `WarmPath.sol`, `KnockoutPath.sol`, `LongPath.sol`, `ColdPath.sol`, `PriceGrid.sol`, `Encoding.sol`, `KnockoutLiq.sol`, `SwapCurve.sol`, `Chaining.sol`, `CurveMath.sol`, `CurveRoll.sol`.
    2. L22 of `CurveRoll.sol`: Duplicate line (`using SafeCast for uint256;`) from L18. Remove one of the two.
    3. L24 of `Chaining.sol`: Duplicate line (`using CurveMath for CurveMath.CurveState;`) from L23. Remove one of the two.

4. Function `KnockoutFlagPath.crossCurveFlag()` always returns `0`. Consider changing the return type of the function or return the value from the call to `crossKnockout()`.

27. For improved readability and maintainability it is advised to use descriptive variable/function/...-names. In this regard, consider changing following instances:
    1. `Directives.PassiveDirective`: The struct defines a unified directive rather than a passive one (Also, there is no `ActiveDirective`)

28. To facilitate logging it is recommended to index address parameters within events. Therefore the `indexed` keyword should be added to the (other) address parameters in
    1. `CrocEvents.ProtocolDividend()`,
    2. `CrocPolicy.CrocGovernAuthority()`,
    3. `CrocPolicy.CrocResolutionOps()`,
    4. `CrocPolicy.CrocResolutionTreasury()`,
    5. `CrocPolicy.CrocEmergencyHalt()`,
    6. `CrocPolicy.CrocPolicySet()`,
    7. `CrocPolicy.CrocPolicyForce()`,
    8. `CrocPolicy.CrocPolicyEmergency()`.

Test Results

Test Suite Results

Initial audit

1. Several tests cover many cases in a function. However, it's unclear which cases are actually covered (edge cases, positive cases, negative cases). Because of that, it's hard to assess each test's quality. For example, the `tick to ratio` test in `TestTickMath.sol` has nine expected statements; unless you analyze the test carefully, it's quite hard to assess what is under test. Consider adopting a more meaningful naming convention (and add comments) so that it's clear what is covered on each test. Moreover, tests with good names are also a great source of documentation; this can be quite helpful for code maintainability.
2. Most of the test cases are positive; they generally only consider the happy-path scenario. However, it's also important to test negative and edge cases (infinity, maximum, minimum). Consider adding more test cases and naming them in a better way.
3. There are many methods across the codebase that require high precision. Make sure to add unit tests proving that the level of precision needed is always met (even on negative cases and other edge cases).
4. Based on the code organization, it's not very clear which files contain unit tests and which ones are integration tests. Consider organizing them in different folders to make it clear what is under test.
5. There are several libraries like `FixedPoint`, `CurveRoll`, `PoolSpecs`, `SafeCast`, amongst others, that do not have unit tests. Consider adding tests for all libraries in the project.
6. `TestEncoding.ts` - The tests in this file only cover positive cases. Consider adding tests to validate that the `Encoding` contract gracefully handles incorrect values or other edge cases correctly.
7. Consider adding tests to each method that can have conduits to validate that those methods will revert if the given conduit is not a contract or non-approved conduit.
8. `TestPool.hotpath.ts` - There is commented code between `L37-L48`. Make sure to remove that code if it's not necessary. Otherwise, uncomment the code and make the necessary adjustments.
9. `TestBitmaps.ts` - Not all the methods in the `Bitmaps` library are tested. Given that this functionality is quite complex, we recommend adding unit tests for each method in the file. Also, add tests for negative cases and other edge cases.
10. Consider adding integration test cases to validate that pools behave correctly in extreme cases (if all liquidity is withdrawn from a pool, if a huge amount of ambient liquidity is added, if a huge amount of concentrated liquidity is added, if a huge or tiny swap is executed, etc.).
11. Throughout the code, there are comments stating assumptions. All of those assumptions need to be validated with tests.
12. `CurveMath.sol` - There are no tests validating the `calcQuoteDelta` method. This method requires a high level of precision, so it needs tests to verify that this requirement is adequately met.
13. `CurveMath.sol` - Consider adding tests to ensure that the inverted flow works well for the different combinations of `isBuy` and `inBaseQty` flags. Also, add tests to ensure that it's never possible for the `denomFlow` to be larger than the `initReserve`.
14. `TestInitPool.ts` - There are no tests to validate negative cases. For example, there are no tests to ensure that it's not possible to initialize a pool that already exists or that it's not possible to create a pool with an uninitialized template.

There is one failing test that has to be fixed.

Fix review update

All tests are executing successfully, and 49 additional tests were added.

```
npx hardhat test
Creating Typechain artifacts in directory typechain for target ethers-v5
Successfully generated Typechain artifacts!


  AgentMask
    ✓ pass relay cond
    ✓ pass relayer origin
    ✓ pass relayer origin
    ✓ fail relayer address
    ✓ fail deadline
    ✓ fail alive
    ✓ fail nonce early
    ✓ fail nonce late
    ✓ fail nonce salt
    ✓ repeat nonce cond (93ms)
    ✓ signature

  BitmapsLib
    ✓ truncateRight
    ✓ truncateLeft
    ✓ isBitSet
    ✓ indexPosLeft
    ✓ indexPosRight
    ✓ shiftPosLeft
```

```
    ✓ shiftPosRight
    ✓ spillPosLeft
    ✓ spillPosRight
    ✓ castIndex
    ✓ lobbyMezzTerm Decomp
    ✓ term shift

TestCompoundMath
    ✓ approx sqrt
    ✓ stack
    ✓ divide
    ✓ shrink
    ✓ price
    ✓ inflate
    ✓ deflate
    ✓ deflate precision

CurveMath
    ✓ active liquidity
    ✓ limit calc
    ✓ limit exhaust qty
    ✓ limit invert
    ✓ limit invert exhaust qty
    ✓ limit infinite
    ✓ limit inifinite invert
    ✓ vig flow
    ✓ vig flow sell
    ✓ vig flow quote denom
    ✓ vig flow limit
    ✓ vig protocol cut
    ✓ vig infinite max
    ✓ roll liq
    ✓ roll liq infinity
    ✓ assimilate liq
    ✓ assimilate liq denom
    ✓ assimilate zero fees
    ✓ assimilate zero liq
    ✓ assimilate one liq
    ✓ assimilate one liq denom
    ✓ assimilate one liq zero fees
    ✓ assimilate one liq denom zero fees
    ✓ derive liq and price flow impact
    ✓ derive liq and price reserve rounding
    ✓ derive liq and price flow entire reserve

Encoding
    ✓ open settlement (46ms)
    ✓ hop settlement
    ✓ hop improve (102ms)
    ✓ pool idx
    ✓ swap
    ✓ ambient (77ms)
    ✓ concentrated (44ms)
    ✓ chain flags (95ms)

TestFixedMath
    ✓ mulQ64
    ✓ mulQ64 Precision
    ✓ mulQ48
    ✓ mulQ48 Precision
    ✓ divQ64
    ✓ divQ64 Precision
    ✓ divQ64Sq Precision
    ✓ recipQ64

Gas Benchmarks Coldpath
    ✓ create pool [@gas-test]
    ✓ mint in virgin pool [@gas-test] (52ms)
    ✓ mint increase liq [@gas-test] (99ms)
    ✓ mint pre-init ticks [@gas-test] (91ms)
    ✓ mint one fresh init [@gas-test] (94ms)
    ✓ mint fresh ticks [@gas-test] (97ms)
    ✓ mint below price [@gas-test] (105ms)
    ✓ mint above price [@gas-test] (91ms)
    ✓ burn partial [@gas-test] (88ms)
    ✓ burn partial level left [@gas-test] (133ms)
    ✓ burn full [@gas-test] (92ms)
    ✓ burn full level left [@gas-test] (134ms)
    ✓ burn outside [@gas-test] (86ms)
    ✓ burn outside left [@gas-test] (122ms)
    ✓ burn liq rewards [@gas-test] (133ms)
    ✓ burn liq level left [@gas-test] (177ms)
    ✓ burn flipped [@gas-test] (209ms)
    ✓ burn flipped level left [@gas-test] (248ms)
    ✓ swap no pre-warm [@gas-test] (95ms)
    ✓ swap small [@gas-test] (138ms)
    ✓ swap tick w/o cross [@gas-test] (138ms)
    ✓ swap bitmap w/o cross [@gas-test] (147ms)
    ✓ swap cross tick [@gas-test] (218ms)
    ✓ swap cross two tick [@gas-test] (253ms)
    ✓ swap cross two tick and bitmap [@gas-test] (257ms)
    ✓ swap cross bitmap between two tick  [@gas-test] (261ms)
    ✓ swap cross many ticks [@gas-test] (514ms)
    ✓ swap cross many bitmap [@gas-test] (153ms)
    ✓ swap surplus [@gas-test] (167ms)
    ✓ mint surplus [@gas-test] (156ms)
    ✓ burn surplus [@gas-test] (195ms)

Gas Benchmarks Native Eth
    ✓ mint in virgin pool [@gas-test] (42ms)
    ✓ mint increase liq [@gas-test] (59ms)
    ✓ mint pre-init ticks [@gas-test] (60ms)
    ✓ mint one fresh init [@gas-test] (60ms)
    ✓ mint fresh ticks [@gas-test] (70ms)
    ✓ mint below price [@gas-test] (58ms)
    ✓ mint above price [@gas-test] (63ms)
    ✓ burn partial [@gas-test] (59ms)
    ✓ burn partial level left [@gas-test] (86ms)
    ✓ burn full [@gas-test] (61ms)
    ✓ burn full level left [@gas-test] (91ms)
    ✓ burn outside [@gas-test] (59ms)
    ✓ burn outside left [@gas-test] (75ms)
    ✓ burn liq rewards [@gas-test] (85ms)
    ✓ burn liq level left [@gas-test] (114ms)
    ✓ burn flipped [@gas-test] (162ms)
    ✓ burn flipped level left [@gas-test] (185ms)
    ✓ harvest fees [@gas-test] (211ms)
    ✓ swap no pre-warm [@gas-test] (66ms)
    ✓ swap small [@gas-test] (105ms)
    ✓ swap small [@gas-test] (92ms)
    ✓ swap small sell [@gas-test] (98ms)
    ✓ swap tick w/o cross [@gas-test] (90ms)
    ✓ swap bitmap w/o cross [@gas-test] (99ms)
    ✓ swap cross tick [@gas-test] (151ms)
    ✓ swap cross two tick [@gas-test] (173ms)
    ✓ swap cross two tick and bitmap [@gas-test] (180ms)
    ✓ swap cross bitmap between two tick  [@gas-test] (190ms)
    ✓ swap cross many ticks [@gas-test] (367ms)
    ✓ swap cross many bitmap [@gas-test] (112ms)

Gas Benchmarks Knockout
    ✓ mint knockout (96ms)
    ✓ mint knockout pre-init pivot (125ms)
    ✓ swap cross full knockout [@gas-test] (145ms)
    ✓ swap cross end of knockout [@gas-test] (167ms)

Gas Benchmarks Proxy Sidecars
    ✓ swap proxy unused [@gas-test] (89ms)
    ✓ swap proxy optimal - forced [@gas-test] (116ms)

Gas Benchmarks
    ✓ mint in virgin pool [@gas-test]
    ✓ mint increase liq [@gas-test] (63ms)
    ✓ mint pre-init ticks [@gas-test] (63ms)
    ✓ mint one fresh init [@gas-test] (65ms)
    ✓ mint fresh ticks [@gas-test] (66ms)
    ✓ mint below price [@gas-test] (64ms)
    ✓ mint above price [@gas-test] (67ms)
    ✓ burn partial [@gas-test] (63ms)
    ✓ burn partial level left [@gas-test] (96ms)
    ✓ burn full [@gas-test] (67ms)
    ✓ burn full level left [@gas-test] (89ms)
    ✓ burn outside [@gas-test] (63ms)
    ✓ burn outside left [@gas-test] (78ms)
    ✓ burn liq rewards [@gas-test] (85ms)
    ✓ burn liq level left [@gas-test] (119ms)
    ✓ burn flipped [@gas-test] (172ms)
```

```
    ✓ burn flipped level left [@gas-test] (192ms)
    ✓ harvest fees [@gas-test] (235ms)
    ✓ swap no pre-warm [@gas-test] (76ms)
    ✓ swap small [@gas-test] (91ms)
    ✓ swap tick w/o cross [@gas-test] (92ms)
    ✓ swap bitmap w/o cross [@gas-test] (108ms)
    ✓ swap cross tick [@gas-test] (143ms)
    ✓ swap cross two tick [@gas-test] (176ms)
    ✓ swap cross two tick and bitmap [@gas-test] (187ms)
    ✓ swap cross bitmap between two tick  [@gas-test] (184ms)
    ✓ swap cross many ticks [@gas-test] (399ms)
    ✓ swap cross many bitmap [@gas-test] (111ms)
    ✓ swap surplus [@gas-test] (118ms)
    ✓ mint surplus [@gas-test] (135ms)
    ✓ burn surplus [@gas-test] (142ms)

  Initialize Pool
    ✓ init token pool (72ms)
    ✓ init ether pool (68ms)

  Knockout Counter Mixin
    ✓ mint position
    ✓ mint ask position
    ✓ mint add pos (43ms)
    ✓ mint pivot stack (54ms)
    ✓ mint pivot arches (40ms)
    ✓ merkle slot pre-warmed
    ✓ burn partial (43ms)
    ✓ burn full (44ms)
    ✓ burn over qty (55ms)
    ✓ burn ask position (54ms)
    ✓ burn bid/ask criss-cross (47ms)
    ✓ burn add pos (56ms)
    ✓ burn pivot stack (62ms)
    ✓ cross position (43ms)
    ✓ cross position sell (43ms)
    ✓ cross multiple (92ms)
    ✓ claim position (43ms)
    ✓ claim multiple pos (70ms)
    ✓ claim stack (168ms)
    ✓ claim before knockout
    ✓ burn after knockout (41ms)
    ✓ bad claim proofs (214ms)
    ✓ recover (162ms)
    ✓ bad recovers (176ms)

  Knockout Liquidity
    ✓ encode pivot
    ✓ encode pos
    ✓ proof no steps
    ✓ proof no steps history
    ✓ proof merkle one step
    ✓ proof merkle multi steps
    ✓ proof merkle middle step
    ✓ assert width
    ✓ assert grid
    ✓ assert disabled
    ✓ assert outside (82ms)
    ✓ assert inside

  LevelBook
    ✓ empty init
    ✓ add fresh liq
    ✓ stack liq
    ✓ add above
    ✓ add below
    ✓ remove partial (38ms)
    ✓ remove full
    ✓ remove over
    ✓ bookmark ticks (55ms)
    ✓ forget ticks (143ms)
    ✓ cross level liq (89ms)
    ✓ cross non level (82ms)
    ✓ odometer add
    ✓ odometer remove partial
    ✓ odometer remove full
    ✓ odometer add/rmove sequence (61ms)
    ✓ above re-clock
    ✓ odometer boundary (38ms)
    ✓ odometer zero init
    ✓ below re-clock
    ✓ cross fee (81ms)
    ✓ cross up (59ms)
    ✓ cross sequence (105ms)

  LiquidityCurve
    ✓ liquidity receive ambient
    ✓ liquidity pay ambient
    ✓ liquidity receive concentrated
    ✓ liquidity pay concentrated
    ✓ multiple pools (70ms)
    ✓ liquidity below range (51ms)
    ✓ liquidity above range (50ms)
    ✓ liquidity below range (50ms)
    ✓ liquidity on lower bump (44ms)
    ✓ liquidity on lower bump wei (46ms)
    ✓ liquidity on upper bump (42ms)
    ✓ liquidity on upper bump wei (42ms)
    ✓ liquidity inside tick (61ms)
    ✓ liquidity rewards

  LiquidityMath
    ✓ add
    ✓ add signed
    ✓ minus

  Pool Conduit
    ✓ mint and burn ambient (58ms)
    ✓ transfer LP token (68ms)
    ✓ no accept concentrated LP (51ms)
    ✓ wrong pool token (384ms)
    ✓ wrong pool index (51ms)

  Sequence Pair
    ✓ two pair sequence (190ms)
    ✓ surplus settle (213ms)
    ✓ surplus partial (228ms)
    ✓ quote entry (194ms)
    ✓ settle mid (200ms)
    ✓ three pair sequence (228ms)

  Sequence Triangle
    ✓ triangle sequence (298ms)
    ✓ triangle arb (405ms)

  Pair
    ✓ two pool arbitrage (240ms)
    ✓ two pool arbitrage quote (236ms)
    ✓ three pools stacked flow (292ms)
    ✓ protocol fee baseline (329ms)
    ✓ protocol fee stack both sides (398ms)
    ✓ protocol fee stack base (422ms)
    ✓ protocol fee stack quote (430ms)
    ✓ pool settings individual (254ms)

  CrocPolicy
    ✓ constructor addresses
    ✓ transfer authority (63ms)
    ✓ authority for transfer authority (89ms)
    ✓ transfer not accepted (112ms)
    ✓ ops resolution
    ✓ ops resolution from treasury
    ✓ ops resolution from emergency
    ✓ ops resolution unauthorized
    ✓ treasury resolution
    ✓ treasury resolution unauthorized
    ✓ emergency halt
    ✓ emergency unauthorized
    ✓ policy invoke
    ✓ policy invoke flag pos
    ✓ policy non conduit
    ✓ policy flag off
    ✓ expired policy
    ✓ set policy unauthorized
    ✓ policy weaken
    ✓ expiry offset (340ms)
    ✓ mandate weaken (39ms)
```

```
    ✓ force weaken flags
    ✓ force weaken mandate
    ✓ force weaken unauthorized
    ✓ emergency policy
    ✓ emergency policy authorized

  Pool Router Agent
    ✓ router agent (43ms)
    ✓ router approve array (43ms)
    ✓ router no cold path
    ✓ router not approved
    ✓ router unnapproved party
    ✓ router unapproved callpath
    ✓ router nonces (463ms)
    ✓ router nonces reset (902ms)

  Pool Relayer Agent
    ✓ relay call (66ms)
    ✓ nonce no repeat (65ms)
    ✓ nonce sequence (85ms)
    ✓ relayer address (57ms)
    ✓ unauthorized relayer (45ms)
    ✓ deadline (48ms)
    ✓ live time condition (44ms)
    ✓ nonce reset (76ms)
    ✓ nonce reset wrong (58ms)
    ✓ nonce reset cond (81ms)
    ✓ nonce reset cond mock args (61ms)
    ✓ reset cond reject (54ms)
    ✓ reset cond bad oracle (56ms)
    ✓ relayer tip (64ms)
    ✓ tip sender (61ms)
    ✓ tip origin (61ms)
    ✓ tip protocol take (80ms)
    ✓ protocol take rate valid

  Pool
    ✓ mint collection (153ms)
    ✓ mint liquidity (206ms)
    ✓ swap simple (121ms)
    ✓ swap protocol fee (148ms)
    ✓ swap sell (127ms)
    ✓ swap sell protocol fee (135ms)
    ✓ swap wrong direction (116ms)
    ✓ swap buy quote output (137ms)
    ✓ swap sell quote output (131ms)
    ✓ swap buy quote proto fee (161ms)
    ✓ swap limit (133ms)
    ✓ swap tick step (256ms)
    ✓ swap tick sell (274ms)
    ✓ swap tick protocol fee (303ms)
    ✓ init protocol fee (570ms)
    ✓ burn payout full (121ms)
    ✓ burn payout sum full (191ms)
    ✓ burn payout tranche (275ms)
    ✓ burn liquidity (335ms)
    ✓ burn payout rewards (447ms)
    ✓ mint blends rewards (490ms)
    ✓ mint ambient (86ms)
    ✓ burn ambient (122ms)
    ✓ mint ambient seed inflator (373ms)
    ✓ burn ambient growth deflator (273ms)
    ✓ burn ambient post growth deflator (352ms)
    ✓ burn ambient over provision (318ms)

  Pool Compound
    ✓ swap->mint (172ms)
    ✓ swap defer (176ms)
    ✓ swap->burn  (173ms)
    ✓ mint concentrated (177ms)
    ✓ multiple range orders (235ms)

  Pool Compound Curve Cache
    ✓ swap curve cache (344ms)

  Pool Conduit
    ✓ mint ambient
    ✓ burn ambient (60ms)
    ✓ mint ambient deflator (154ms)
    ✓ mint concentrated (40ms)
    ✓ burn concentrated (71ms)
    ✓ mint concentrated deflator (165ms)
    ✓ mint reject (44ms)
    ✓ burn reject (108ms)

  Pool Surplus Deposits
    ✓ deposit
    ✓ deposit native (64ms)
    ✓ deposit native insufficient value
    ✓ deposit permit (44ms)
    ✓ disburse
    ✓ disburse native (73ms)
    ✓ disburse full
    ✓ disburse full infer
    ✓ disburse over-size
    ✓ disburse all but
    ✓ transfer
    ✓ transfer full
    ✓ transfer all but
    ✓ transfer full infer
    ✓ transfer over
    ✓ side pocket (38ms)
    ✓ side partial (40ms)
    ✓ side zero full (50ms)
    ✓ side all but (45ms)
    ✓ side pocket protects capital (55ms)

  Pool Ethereum
    ✓ mint (46ms)
    ✓ balance client side (48ms)
    ✓ burn (86ms)
    ✓ mint ambient
    ✓ burn ambient (76ms)
    ✓ swap protocol fee (171ms)

  Pool Ethereum Hotpath
    ✓ mint
    ✓ balance client side
    ✓ burn (65ms)
    ✓ mint ambient
    ✓ burn ambient (53ms)
    ✓ swap protocol fee (123ms)

  Pool Governance
    ✓ transfer authority (118ms)
    ✓ set treasury (230ms)
    ✓ collect treasury (251ms)
    ✓ collect treasury time delay (231ms)
    ✓ safe mode (235ms)
    ✓ init liq valid bounds
    ✓ take rate (70ms)

  Pool Grid
    ✓ grid required (253ms)
    ✓ grid required hotpath (208ms)
    ✓ grid revised (95ms)
    ✓ burn after revised (189ms)
    ✓ burn after revised hotpath (167ms)
    ✓ price improve - no settings (107ms)
    ✓ price improve - settings (98ms)
    ✓ price improve burn full (115ms)
    ✓ price improve burn partial (66ms)
    ✓ price improve burn hot path (118ms)
    ✓ price improve - quote side (97ms)
    ✓ price improve - away (103ms)
    ✓ price improve - wrong base side
    ✓ price improve - wrong quote side

  Pool Harvest
    ✓ harvest rewards (127ms)
    ✓ harvest deplete (145ms)
    ✓ burn deplete (177ms)
    ✓ harvest re-fill (199ms)

  Pool HotPath
    ✓ mint collection (106ms)
    ✓ mint liquidity (142ms)
```

```
    ✓ swap simple (106ms)
    ✓ swap protocol fee (124ms)
    ✓ swap sell (86ms)
    ✓ swap sell protocol fee (121ms)
    ✓ swap wrong direction (68ms)
    ✓ swap buy quote output (95ms)
    ✓ swap sell quote output (105ms)
    ✓ swap buy quote proto fee (129ms)
    ✓ swap limit (97ms)
    ✓ swap tick step (195ms)
    ✓ swap tick sell (215ms)
    ✓ swap tick protocol fee (245ms)
    ✓ burn payout full (92ms)
    ✓ burn payout sum full (138ms)
    ✓ burn payout tranche (199ms)
    ✓ burn liquidity (247ms)
    ✓ burn payout rewards (301ms)
    ✓ mint blends rewards (364ms)
    ✓ mint ambient (72ms)
    ✓ burn ambient (85ms)
    ✓ mint ambient seed inflator (283ms)
    ✓ burn ambient growth deflator (216ms)
    ✓ burn ambient post growth deflator (304ms)
    ✓ burn ambient over provision (266ms)

  Pool JIT
    ✓ jit window (112ms)
    ✓ mint in window (150ms)
    ✓ jit window too large (95ms)

  Pool Knockout Liq
    ✓ mint flow (77ms)
    ✓ mint flow ask (68ms)
    ✓ mint off-grid (75ms)
    ✓ mint bad width (125ms)
    ✓ mint inside mid (119ms)
    ✓ mint bad inside mid (61ms)
    ✓ burn partial (102ms)
    ✓ burn full liq (99ms)
    ✓ swap into active range (331ms)
    ✓ swap into active range ask (324ms)
    ✓ swap knockout (352ms)
    ✓ swap knockout ask (366ms)
    ✓ claim knockout (361ms)
    ✓ claim knockout ask (378ms)
    ✓ claim knockout proof (951ms)
    ✓ bad proof (968ms)
    ✓ recover knockout (367ms)
    ✓ claim knockout twice (398ms)
    ✓ recover knockout twice (410ms)
    ✓ knockout no repeat (594ms)

  Pool Warm LP Path
    ✓ mint ambient base
    ✓ mint ambient quote
    ✓ burn ambient base (60ms)
    ✓ burn ambient quote (56ms)
    ✓ mint conc base (44ms)
    ✓ mint conc qutoe (40ms)
    ✓ burn conc base (73ms)
    ✓ burn conc quoe (71ms)
    ✓ out of range base (43ms)
    ✓ out of range quote (38ms)

  permissioned pool
    ✓ permit oracle (160ms)
    ✓ mint/burn concentrated (129ms)
    ✓ mint/burn ambient (77ms)
    ✓ compound directive (152ms)

  Pool Proxy Paths
    ✓ swap no proxy (45ms)
    ✓ swap proxy (124ms)
    ✓ swap proxy optional (138ms)
    ✓ swap force proxy (71ms)
    ✓ swap long path okay (167ms)
    ✓ cannot upgrade boot path (54ms)
    ✓ upgrade requires contract address (63ms)
    ✓ requires proxy contract accept (73ms)

  Pool Rebalance
    ✓ rebalance range (207ms)
    ✓ rebalance gas (146ms)
    ✓ rebalance liq (193ms)
    ✓ rebalance liq gas [@gas-test] (145ms)

  Pool Security
    ✓ double initialize (175ms)
    ✓ template disabled (54ms)
    ✓ pre-initialize (96ms)
    ✓ mint outside tick range (195ms)
    ✓ over burn (518ms)
    ✓ burn steal (551ms)

  Pool Surplus
    ✓ balance and withdraw (79ms)
    ✓ debit entry (59ms)
    ✓ debit partial entry (82ms)
    ✓ credit entry (114ms)
    ✓ debit exit (57ms)
    ✓ debit partial exit (66ms)
    ✓ credit exit (100ms)
    ✓ swap hotpath (93ms)
    ✓ mint hotpath (77ms)
    ✓ burn hotpath (109ms)
    ✓ mint ambient hotpath (69ms)
    ✓ burn ambient hotpath (87ms)
    ✓ swap base settle (78ms)
    ✓ swap quote settle (71ms)
    ✓ mint base settle (68ms)
    ✓ mint quote settle (70ms)

  Pool Surplus Ether
    ✓ balance and withdraw (53ms)

  PositionRegistrar
    ✓ empty init
    ✓ add liq
    ✓ add stack
    ✓ add multi pos (42ms)
    ✓ burn partial
    ✓ burn full
    ✓ burn position only
    ✓ burn rewards (57ms)
    ✓ transfer position
    ✓ transfer collision

  Price Improve
    ✓ ticks out of order
    ✓ non-improvable
    ✓ clip inside positive
    ✓ clip inside negative
    ✓ clip inside over zero
    ✓ clip below
    ✓ clip above
    ✓ unit tick (43ms)
    ✓ scale thresh
    ✓ grid pin ask wings
    ✓ grid pin bid wings
    ✓ grid pin wings both sides
    ✓ on grid
    ✓ verify (56ms)

  Protocol Account
    ✓ accum
    ✓ disburse
    ✓ ethereum token
    ✓ disburse post

  Query Impact
    ✓ small buy (88ms)
    ✓ small sell (86ms)
    ✓ buy denom (84ms)
    ✓ sell denom (87ms)
    ✓ large buy (454ms)
    ✓ large sell (477ms)
    ✓ bump ticks (531ms)
```

```
    ✓ multiple bump ticks (692ms)

  Roll between pairs
    ✓ two pairs (279ms)
    ✓ mint -> swap (188ms)
    ✓ quote entry (197ms)
    ✓ three pair sequence (241ms)

  Pair roll triangle
    ✓ triangle arb (429ms)
    ✓ roll surplus (309ms)
    ✓ roll surplus diff sides (312ms)

  Roll Pools
    ✓ roll between pools (282ms)
    ✓ roll on exit token (277ms)
    ✓ different sides pools (268ms)

  Roll Surplus
    ✓ roll surplus (87ms)
    ✓ roll stacked (105ms)
    ✓ surplus exit (89ms)
    ✓ surplus entry+exit (97ms)

  Rolling Back Fill
    ✓ swap->mint ambient (101ms)
    ✓ ambient seed deflator (356ms)
    ✓ exit at base (98ms)
    ✓ swap quote->mint ambient (102ms)
    ✓ entry at quote (105ms)
    ✓ swap->burn ambient (102ms)
    ✓ mint ambient -> swap (102ms)
    ✓ burn ambient -> swap (105ms)
    ✓ swap roll flip direction (103ms)
    ✓ swap roll flip direction reverse (104ms)
    ✓ swap->mint range (117ms)
    ✓ swap->burn range (118ms)
    ✓ quote -> mint range (121ms)
    ✓ swap->mint below range (115ms)
    ✓ quote -> mint below range (114ms)
    ✓ swap -> mint wrong side (126ms)
    ✓ reposition range (125ms)

  Settle Layer Ethereum
    ✓ settle debit (108ms)
    ✓ settle credit (96ms)
    ✓ settle flat (88ms)
    ✓ settle final on non-ethereum token (99ms)
    ✓ settle debit surplus (127ms)
    ✓ settle debit surplus partial (139ms)
    ✓ settle credit (115ms)
    ✓ settle refund (110ms)

  Settle Layer
    ✓ debit
    ✓ credit
    ✓ debit shortfall
    ✓ credit shortfall
    ✓ zero
    ✓ debit ether
    ✓ credit ether
    ✓ debit ether shortfall
    ✓ debit ether overpay
    ✓ debit ether overpay surplus
    ✓ debit ether double spend
    ✓ credit ether shortfall
    ✓ credit surplus
    ✓ debit surplus (45ms)
    ✓ limit qty credit
    ✓ limit qty debit
    ✓ limit sign (50ms)
    ✓ dust
    ✓ no dust on debit

  Swap Curve
    ✓ swap full qty
    ✓ swap fee full qty
    ✓ swap fee+proto full qty (41ms)
    ✓ swap paid cumulative
    ✓ swap sell (39ms)
    ✓ swap sell fee
    ✓ swap quote denom
    ✓ swap quote denom fees (39ms)
    ✓ swap quote sell
    ✓ swap quote sell fee (42ms)
    ✓ swap bump price
    ✓ swap bump sell
    ✓ swap bump denom
    ✓ swap limit price
    ✓ swap limit fee
    ✓ swap limit sell
    ✓ swap bump infinity
    ✓ swap bump sell infinity
    ✓ swap infinity
    ✓ swap infinity sell
    ✓ swap infinity quote
    ✓ swap infinity quote sell
    ✓ swap zero liq base buy
    ✓ swap zero liq quote buy
    ✓ swap zero liq base sell
    ✓ swap zero liq quote buy

  TickCensus
    ✓ empty bitmap (143ms)
    ✓ bookmark tick
    ✓ bookmark repeat
    ✓ bookmark two shared
    ✓ bookmark multiple across
    ✓ forget reset
    ✓ forget repeat
    ✓ forget shared
    ✓ forget multiple across (45ms)
    ✓ pin buy
    ✓ pin sell
    ✓ pin sell at
    ✓ pin edge
    ✓ pin edge barrier
    ✓ pin buy spill
    ✓ pin sell spill
    ✓ pin buy zero point
    ✓ pin sell zero point
    ✓ seek empty (137ms)
    ✓ seek over cliff (168ms)
    ✓ seek terminus neighbor (38ms)
    ✓ seek immediate neighbor (54ms)
    ✓ seek through mezz (45ms)
    ✓ seek immediate mezz (43ms)
    ✓ seek mezz caged (42ms)
    ✓ seek mezz inner caged (67ms)
    ✓ seek through lobby (47ms)
    ✓ seek lobby lookback
    ✓ seek lobby lookback reverse

  Tick Math
    ✓ tick to ratio
    ✓ ratio to tick
    ✓ min tick
    ✓ max tick
    ✓ outside bounds

  Token Precision
    ✓ base token medium low liquidity
    ✓ base token low liquidity
    ✓ base token very low liquidity
    ✓ base token medium liquidity
    ✓ base token medium high liquidity
    ✓ base token high liquidity
    ✓ base token very high liquidity
    ✓ quote token medium low liquidity
    ✓ quote token low liquidity
    ✓ quote token very low liquidity
    ✓ quote token medium liquidity
    ✓ quote token medium high liquidity
    ✓ quote token high liquidity
    ✓ quote token very high liquidity
```

```
  686 passing (4m)
```

Code Coverage

Initial audit

Quantstamp usually recommends developers increase the branch coverage to `90%` and above before a project goes live in order to avoid hidden functional bugs that might not be easy to spot during the development phase. For branch code coverage, the current targeted files by the audit achieve an efficient score that can be improved further.

Fix review update

The generated branch coverage still does not meet the 90% threshold. Please note that the following tests are failing:

```
586 passing (7m)
  3 failing

  1) Gas Benchmarks Knockout
       mint knockout:
     AssertionError: Expected "242578" to be less than 228000

  2) Gas Benchmarks Knockout
       mint knockout pre-init pivot:
     AssertionError: Expected "133066" to be less than 106000


  3) Pool Rebalance
       rebalance gas:
     AssertionError: Expected "405479" to be less than 315000
```

Initial audit

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
CrocEvents.sol	100	100	100	100	
CrocSwapDex.sol	100	100	100	100	
contracts/callpaths/	99.42	92.59	100	99.34	
ColdPath.sol	100	96.15	100	100	
HotPath.sol	95.83	50	100	95.83	87
KnockoutPath.sol	100	83.33	100	100	
LongPath.sol	100	100	100	100	
MicroPaths.sol	100	100	100	100	
MultiPath.sol	100	87.5	100	100	
SafeModePath.sol	100	100	100	100	
WarmPath.sol	98.48	96.43	100	98.11	114
contracts/governance/	100	100	100	100	
CrocPolicy.sol	100	100	100	100	
contracts/interfaces/	100	100	100	100	
ICrocCondOracle.sol	100	100	100	100	
ICrocLpConduit.sol	100	100	100	100	
ICrocMinion.sol	100	100	100	100	
ICrocPermitOracle.sol	100	100	100	100	
ICrocVirtualToken.sol	100	100	100	100	
IERC20Minimal.sol	100	100	100	100	
contracts/libraries/	98.6	90.85	97.84	98.77	
BitMath.sol	100	100	100	100	
Bitmaps.sol	100	100	100	100	
Chaining.sol	95.45	88.24	100	98.25	405
CompoundMath.sol	100	50	100	100	
CurveAssimilate.sol	100	100	100	100	
CurveCache.sol	100	100	100	100	
CurveMath.sol	100	100	100	100	
CurveRoll.sol	**98.36**	**92.31**	**100**	**100**	
Directives.sol	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
Encoding.sol	100	100	82.35	82.35	155,169,231
FixedPoint.sol	100	100	100	100	
KnockoutLiq.sol	100	90	100	100	
LiquidityMath.sol	91.67	83.33	100	100	
PoolSpecs.sol	100	50	100	100	
PriceGrid.sol	100	96.67	100	100	
ProtocolCmd.sol	100	100	100	100	
SafeCast.sol	100	100	83.33	83.33	11
SwapCurve.sol	100	83.33	100	100	
TickMath.sol	100	100	100	100	
TokenFlow.sol	100	100	100	100	
TransferHelper.sol	100	100	100	100	
contracts/mixins/	98.91	89.51	100	99.18	
AgentMask.sol	96	80.95	100	97.44	267,327
ColdInjector.sol	100	83.33	100	100	
DepositDesk.sol	100	100	100	100	
KnockoutCounter.sol	100	90.91	100	100	
LevelBook.sol	100	100	100	100	
LiquidityCurve.sol	100	80.77	100	100	
MarketSequencer.sol	90.74	80	100	92.59	318,319,320,321
PoolRegistry.sol	100	88.24	100	100	
PositionRegistrar.sol	100	100	100	100	
ProtocolAccount.sol	100	83.33	100	100	
SettleLayer.sol	100	95.24	100	100	
StorageLayout.sol	100	100	100	100	
TickCensus.sol	100	100	100	100	
TradeMatcher.sol	100	100	100	100	
contracts/periphery/	100	87.5	100	100	
CrocLpErc20.sol	100	87.5	100	100	
All files	**99.01**	**90.71**	**99.19**	**99.15**	

Fix review update

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
CrocEvents.sol	100	100	100	100	
CrocSwapDex.sol	100	100	100	100	
contracts/callpaths/	94.6	89.83	86.3	94.3	
BootPath.sol	80	83.33	50	80	43,48,75
ColdPath.sol	94.21	92.86	92.59	94.06	… 313,315,336
HotPath.sol	96.3	50	100	96.3	94
KnockoutPath.sol	93.02	83.33	77.78	92.5	46,99,189
LongPath.sol	97.14	100	75	97.14	109
MicroPaths.sol	97.56	100	83.33	97.56	208
SafeModePath.sol	66.67	100	66.67	66.67	25
WarmPath.sol	**97.01**	**96.43**	**92.31**	**96.3**	**111,298**
contracts/governance/	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
CrocPolicy.sol	100	100	100	100	
contracts/interfaces/	100	100	100	100	
ICrocCondOracle.sol	100	100	100	100	
ICrocLpConduit.sol	100	100	100	100	
ICrocMinion.sol	100	100	100	100	
ICrocPermitOracle.sol	100	100	100	100	
ICrocVirtualToken.sol	100	100	100	100	
IERC20Minimal.sol	100	100	100	100	
contracts/libraries/	98.77	90.54	99.42	99.54	
BitMath.sol	100	100	100	100	
Bitmaps.sol	100	100	100	100	
Chaining.sol	95.52	88.24	100	98.28	431
CompoundMath.sol	100	75	100	100	
CurveAssimilate.sol	100	100	100	100	
CurveCache.sol	100	100	100	100	
CurveMath.sol	100	100	100	100	
CurveRoll.sol	98.36	92.31	100	100	
Directives.sol	100	100	100	100	
Encoding.sol	100	50	100	100	
FixedPoint.sol	100	100	100	100	
KnockoutLiq.sol	100	87.5	100	100	
LiquidityMath.sol	96	90	100	100	
PoolSpecs.sol	100	50	100	100	
PriceGrid.sol	100	96.67	100	100	
ProtocolCmd.sol	100	100	100	100	
SafeCast.sol	100	100	83.33	83.33	11
SwapCurve.sol	100	66.67	100	100	
TickMath.sol	100	100	100	100	
TokenFlow.sol	100	100	100	100	
TransferHelper.sol	100	100	100	100	
contracts/mixins/	98.31	88.08	98.64	98.56	
AgentMask.sol	93.9	77.27	92.31	95.29	249,251,311,363
DepositDesk.sol	100	100	100	100	
KnockoutCounter.sol	100	93.33	100	100	
LevelBook.sol	100	100	100	100	
LiquidityCurve.sol	100	80.77	100	100	
MarketSequencer.sol	90.91	80	100	92.59	315,316,317,318
PoolRegistry.sol	100	84.78	100	100	
PositionRegistrar.sol	100	100	100	100	
ProtocolAccount.sol	100	83.33	100	100	
ProxyCaller.sol	91.89	81.82	90.91	91.89	41,42,44
SettleLayer.sol	100	95.24	100	100	
StorageLayout.sol	100	100	100	100	
TickCensus.sol	100	100	100	100	
TradeMatcher.sol	**100**	**95**	**100**	**100**	
contracts/periphery/	100	87.5	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
CrocLpErc20.sol	100	87.5	100	100	
contracts/vendor/compound/	0	0	0	0	
Timelock.sol	0	0	0	0	… 104,106,111
All files	**95.28**	**84.78**	**95.45**	**95.62**	

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

```
7bed4fe9fd24261f3a96ebbad618a67337fe8b5c4745be4df8962b217e57aff3 ./contracts/CrocSwapDex.sol
ef78fb7ac50c20226c3e8dd0f51f8fd92e0a5858f798327556c0c50b5fad305f ./contracts/CrocEvents.sol
c0b6c83211730a5a8a3af0b60d43850a51e540baf834b8df11bfb4106b4a38e4 ./contracts/callpaths/KnockoutPath.sol
bf3b28192f30ca1d852e3a1afd6d10e8005861c4742377b2a8a88b927e07f846 ./contracts/callpaths/SafeModePath.sol
6258f4e1d758c688e015551c0fcbd5fdb46b72014e5d084faaf89f922931d806 ./contracts/callpaths/MultiPath.sol
91342504f8c6ee28054063ea840e98ed361cbf7ac23c452c31cbb55b8711bff8 ./contracts/callpaths/LongPath.sol
228a453a4740e0f2794b5d55594c2c55063f856e9dd252f0132b24819b79412f ./contracts/callpaths/WarmPath.sol
41f0d19d6c0a187b79959bec6c9374082c1a17fdc34539dc8d7fa0d99fbe0bd3 ./contracts/callpaths/HotPath.sol
55abdcd57e71cad63f1373e0c57420e3be51bf2ec8350fa0407fd2f09811ebeb ./contracts/callpaths/ColdPath.sol
1976538419c872f7c5b7fb46ff410f9f82398b9418349d509a7f15d77f881ef8 ./contracts/callpaths/MicroPaths.sol
820dc253dfd176ede0811eaa6cc7e30e272dd1aba6fba164e379e9a432d221d4 ./contracts/interfaces/ICrocVirtualToken.sol
62ca5ebf2bd939df27bf45743b732cafb48bf826be5ccd6fecc0fcd9f7291148 ./contracts/interfaces/IERC20Minimal.sol
a280c7d3d6f6c13e3043b6604bc7734e1228aeafa0aacfc53c277e54267cff29 ./contracts/interfaces/ICrocCondOracle.sol
44ba0543eb7f3026dcd19c9a871941edfef90854d64a4c043b723bcfc0e1a127 ./contracts/interfaces/ICrocLpConduit.sol
208e38497f00704da2f7689fb68cc2e11186e1a3e3bbd11c7acce05e43a40766 ./contracts/interfaces/ICrocPermitOracle.sol
c28076149fd6b54a7825dfef5238648aa2e3ad43aba631d586908c63a5b731bd ./contracts/interfaces/ICrocMinion.sol
164ce21e93b3ff28bfa6aaf89c4a9ae4454ec76db735e3deee97ad44c90f7f9b ./contracts/lens/CrocQuery.sol
ab7d7d4aaba5365c70f16834b675ef3a4a6dae7f24ea94dd7fae9528ea06af1b ./contracts/periphery/CrocLpErc20.sol
883cd30029d8026828584c9a99cce67e80a90532360f946edbfd6b078b785b23 ./contracts/governance/CrocPolicy.sol
ef694c18a2c0f218ef2bec16a452c0c7441ac367d8ce239e6d8f6ed6ebef9cd1 ./contracts/mixins/TickCensus.sol
b61e8899a5ae3daec54e3348fb7f8c4f6a7351bda89c605ed2d35f4cadadee41 ./contracts/mixins/PoolRegistry.sol
1fd59ce9cb4503d5e703393e057f989fdab6df02fe5b2c13b6fd758a4239964a ./contracts/mixins/StorageLayout.sol
cc127c54260b603365bf9a2b80fda6ac36e6c211bcdd4ec91d5ad7e1ce058cec ./contracts/mixins/DepositDesk.sol
5d8241e84f910bbba7a1951ba8898430daa0639840c40b8139fd5f9283c9201b ./contracts/mixins/PositionRegistrar.sol
ac027eaf5bb4f1f50ec4622f8005e1767552021df38abe8ad8a9d7a8aa66d7fe ./contracts/mixins/TradeMatcher.sol
5ef9d83d11c98b8408d0c0dd2c2a3bb1e8e9e1b095ed35256d58ea957f7661e4 ./contracts/mixins/LevelBook.sol
32f37480cf02c403dc77093b121e6c75d3355abedaa205961c9666b3a766f05a ./contracts/mixins/KnockoutCounter.sol
3d1358a65bd65ae8bdd9d471e20a04e41b55556373b0267f9be984a6d74475d1 ./contracts/mixins/ColdInjector.sol
603c948aabfed4d8f14cea64b9259a454953f1cfd4cf7510e8b931bd7ef0dea2 ./contracts/mixins/AgentMask.sol
ede416e50cfc67dcdf7e728418243e294f8dd00b2b96ffef4e341f7e48ec82f5 ./contracts/mixins/SettleLayer.sol
0d81fa29255aa3fdd6c91980173c70a00d8e2aa89917f5ecf182018b2c09efbc ./contracts/mixins/LiquidityCurve.sol
ebc1937172208ed5d9a3bbeafc4af63775042d6b13e47622a08ac91a516b7153 ./contracts/mixins/ProtocolAccount.sol
9c4ed90be2cad1594dfd3ac4c85ea79549086267ea696fdd63a040017f63032a ./contracts/mixins/MarketSequencer.sol
410fc58a5dffd676332e4f20ed66be8e5014721b4c17bb8d8875a0f45eb6dfee ./contracts/libraries/TokenFlow.sol
6a3469923cf33cbea9a4a60e322f33e2de9fdf604fccda07ecddd3f97b48e59a ./contracts/libraries/Directives.sol
484a16cb6b1f6e59669308928447980ca523953683439902bbc490b9f56b8856 ./contracts/libraries/FixedPoint.sol
bb84edd3e8ac6423d5f91045bfc3efaa72d3e4902156a310384ed2b6ff068d34 ./contracts/libraries/CurveCache.sol
b068a334eae3fe888aee761f15f605d48e4be1a862c72fd0b2e565db33fd28b8 ./contracts/libraries/SwapCurve.sol
9a00c7df62f3421e80baa2a8e6009bc8132e590881b8840bd4384c849ffd9652 ./contracts/libraries/Bitmaps.sol
2943eeb8813c6c1c0538e435d8212a3a31d928a55b434dbd7b57ea836eb9b5a4 ./contracts/libraries/Chaining.sol
d7225851b16b93d108891e053d4fce88b87978b89fb4f62bf875ae69d12254b7 ./contracts/libraries/CurveAssimilate.sol
6230e68a2a28b2912f077c592fef2cc3eadb8275c480f592ce51a1b66c513cd0 ./contracts/libraries/LiquidityMath.sol
5129d61095875284dbd81f710c605a110344a81245ec6bdf1b4c64711aee0768 ./contracts/libraries/Encoding.sol
c343b5f30228a6ea6013f71a42ec9eb1a08dc3b1ba2522e77940b55fd68e8bba ./contracts/libraries/TickMath.sol
ebc7396d8a6ba68d2bc4a85f9c968709f3874a3ac54c84b86f0018c03bf86943 ./contracts/libraries/ProtocolCmd.sol
c723bbe9547f64513627f290aa583d3a8d568f13cd5425729b7468f15b0f4753 ./contracts/libraries/PoolSpecs.sol
066e07f9f9bb0cd3a2d190884dc108dc975583e83d08926a84f2d5743b2a5864 ./contracts/libraries/TransferHelper.sol
```

e0792bda45fb7ea18bc6736354e3fac0c70465ebaba9d73e8b323ab5a0bfe1c3 ./contracts/libraries/CurveRoll.sol

9b1bdf94e5e4e0ee45df83c978c7bdd016bdd222154938ad2cfacb0d9d2e256e ./contracts/libraries/KnockoutLiq.sol

d8d7ce7808729a90b4e04ac2c417fdd308dfd427e787a8d6f974f449289b295e ./contracts/libraries/CompoundMath.sol

927dd011fd44e594edfbeba48382d9e88fe21d47584b999e14570e8c74260769 ./contracts/libraries/PriceGrid.sol

767652ab0f4fab6b088cffd46bdc3045e5b7974b7637e3ee0ce936fe0267afdf ./contracts/libraries/BitMath.sol

de6b13446c1a908893dff554175a0d393ddfa5bd23f9578c42bb1eb2f3fc1bc8 ./contracts/libraries/CurveMath.sol

72bad5d86c3f41bad2062232fd78b7acda2ecc14891aa5bcd5b42e98374e4eb5 ./contracts/libraries/SafeCast.sol

Tests

8585dc76d3f498a9b36d65695117a7bb2cf9f473062d697aac9ef6f96848a12d ./test/TestGas.cold.ts

23ecb7f7eb40c67169a6a5a0018f56d4fbb6f711545a995a4e7213e6db2e6f27 ./test/TestPool.agent.ts

4ca7a1164267909232a8541129566f4b253c324cbee394341c811023562803fd ./test/TestLiquidityMath.ts

3fe55d4d73578be41c7c64cec80b8fd6b78ce5c3ecf4c2e5812916525432643f ./test/EncodeSimple.ts

6ed4e219f61e6f6f676b07090f87cbf33d18de54ca55f690026c0d1b8b7cc069 ./test/TestPool.conduit.ts

1f46c2b0f7bf9413f1daa510edca84f6d6f2376ceaa519aa11ece3525105dcdb ./test/TestKnockoutCounter.ts

1b3928948810f566d8cee6fd91abbe79ebb59d297f600571ce9c54918ddbd353 ./test/TestPool.sec.ts

ccaf4a9170dde859f2fe7d8cb2837f470af54bb4a80a3dd3c941e94c048d204e ./test/TestPool.hotpath.ts

0ea2857313f7d6906bdf5a37284aacb65a730d89ad0dc3c0d8fa0691d4d3c447 ./test/TestRoll.ts

e48b32d6772b47cf2ebc17cdb3216edd130b010adf1b264d4de8cc02145f0f8d ./test/EncodeOrder.ts

4d3f0f91b4f237fa511f7c4f21950113f8d2f4bb826caf953cafddfe4ee5fdf2 ./test/TestPool.swap.ts

025d55c084979806dc5b7d9192b6f6b138f19bbbfc6563288afd63d83cd9af82 ./test/FixedPoint.ts

6b86164ce105b6bd1455087c4d4a794b76e21383693d99fca067499dd5077bb3 ./test/TestPositionRegistrar.ts

ec0e68d71c90193f7117331237b2cc39a2debd226b4b85d36f463201e36b76d1 ./test/TestPool.surplus.ts

de1d7be8329a05406cd79175bf8bbb285f10589f5a70d32da0b046a4d79ff207 ./test/TestPool.proxy.ts

581500b74de632d4afff7c6b64e94288cee6aad47f7e2a7d7abcdddf1fb18e6c ./test/TestPool.jit.ts

712e88103cb39447fc749db2cdddb73fd377c34b65fd821ec88a635f6fc14b0e ./test/TestLpErc20.ts

b48196cd559410198fe50fc3572b3da1ba2663c34dda5dffc2ad2733d1554b0e ./test/TestPool.govern.ts

7475623149d3ee3101457540d385f4b0e975992426fb4e942ec52f83ebb01b45 ./test/TestPriceGrid.ts

2c116b982235a054ba4f41a842e9b9ae8a0b9087df74083a0e8a003cde993348 ./test/TestPool.knockout.ts

e4d6ab404e0703d65a6a0f6ec39eb8ffe270358eac2ad24c59b471b59d01f93f ./test/TestLevelBook.ts

f4e1ee8088bb861825a0c2e752c9ee871b545a8138557bfb8094226483f9090b ./test/TestPool.rebal.ts

29604e15f16bf05dd0cb557bd2f1e8d7407c1dff1ccf97e146e195dfd7fca552 ./test/TestLiqCurve.ts

97115e992fc2dbd0189a959aefce3d2d66c5ba03063a5569f6e9d4871a411f25 ./test/TestSwapCurve.ts

8fbdf09fe0f02448af116e8a91738c36e3d79d0264d1827c67f2d20fa93c1ecf ./test/TestCompoundMath.ts

16d4eaee60528a47d97c7b0b24061b6a37f245485f486323871624f7e2098eb5 ./test/TestPair.ts

ce12a7e1fbf6b22723fc522f9d5dc4511b94ffa735cc4f0b3aaadb8c880b8715 ./test/TestCurveMath.ts

30e2a46f4ca3bbc04a2271dd41c9b8236b4756baec4fc7cf182659b6662e8059 ./test/TestRoll.pairs.ts

f1c5bfa026a0f6407689d2719054709278553a6d009b288482cfd8ba4d77eabf ./test/TestRoll.pools.ts

0e9d0da5b8dfe26a7ad336b3e6647c185f2803fc032fbdfa9457307e4f1b63a7 ./test/TestGas.ts

393e814c9effb957266a311311eb30e3fd2a3635cef7845ebac22fa4ff101241 ./test/TestPolicy.ts

d763df5ccc0038f17b746709c0f507b1ee0f8b0dfd9c48a5f94d810b0fa6f94e ./test/TestPool.deposit.ts

0d84304896cb012f2366368d3a12f337680c545b7bfd1ac09d4648d2084ccdf8 ./test/TestPool.basic.ts

e30b921654a752e3d7453af589cd107e7e891745625d0f28abd172c7ff598662 ./test/TestPool.grid.ts

f781ceca7411ba27512e9cb50772fc370cf3097cddcc4baa4386f31783b52b34 ./test/TestPool.eth.ts

b8d091dbd6c068d409e6eb8c935aa79bd15039cb40dbbffaec17309df7e4adb7 ./test/TestGas.proxy.ts

3a4df9250b1c2e9d1aa9a7bfa0e6f33c6a78d7efd67b4b07bf78a7d8f9ec7407 ./test/TestTickMath.ts

a8bb8ba0ecd85b051345a34633a3d199255206bfc6e1f021ee1a88e2690f527d ./test/TestInitPool.ts

cecd79d56c9689e531f7681c4e6fd11d3adac5f979986a8c13bb158f12e27838 ./test/TestRoll.surplus.ts

267196f4ed9ec8767881c7c37db766742cbf8fdd5f44e678b900c82405d9e028 ./test/TestPool.lp.ts

7c5ccd7124dc131855ad9a4303647fd196ed0fe5343427e961278cf1fd808b55 ./test/TestPool.multicall.ts

fe932f27a31189c61601b821803a4cc7c48a852d0d8271ab3caf9ebc5d0dc537 ./test/TestProtocolAcct.ts

6c5f426b5985bfb7c40fc4c8c308014f58e695eb2bf8489fecd70c9ed9feebe7 ./test/TestPool.extr.ts

106534b281be6e7874b40e739af8744282b891e17e2b7f05bb100ce53b4ea77a ./test/TestGas.knockout.ts

9332e246c4cb248e877eb4ee5a5a6d1dcfea553cd0b8b7ae58645b005544f106 ./test/TestGas.comp.ts

fc0a0eb3f46df98f333851cfb89016f432c75879d926efdb9caecd5693760e64 ./test/TestPool.comp.ts

056a7b4c81603a8e2775f047e7e1e0066eb06785d187d100d9dfb2dad432dc51 ./test/TestTickCensus.ts

d96b7e3773b52c09a05bbce8b49b46145c7277c309690a405871bd28b4d216af ./test/TestPair.seq.ts

a9cb81d3872f331d4e4c2e775e10dc56509f9fae33ef1a87e0d8bfd86721ebcd ./test/TestPool.virt.ts

fbe5549b46bb85157074d3abe79da7cc2193335873bb11d516bafb9f4545d2d7 ./test/TestPool.harvest.ts

4c42179ca7b702ce0d260769c884f8de1647aff879ac2cabc3f218e07f4eaf29 ./test/TestKnockoutLiq.ts

```
530a5d4a07123821f165e5f3bd4f1d03f1e79a9a60ea196526f42f272581857a ./test/TestAgentMask.ts
d0854dff3475730d8b703015a0c45ca965cc4dab1c5dd05a0d797543d145cb55 ./test/TestPool.permit.ts
012afedbf790b2d2011f0acf0fd45921c1b124cfc72da12996eef70ec81459c8 ./test/TestBitmaps.ts
d0cd865bcf2e3ba4eb9421686bc071dc70a8f4b7e59e45dfc08c6dd9fc6caa04 ./test/TestSettle.eth.ts
c8c26ac54eb98fd45eab7fd081e75ffae9769aa99d8d090139dbc3f341c58eea ./test/FacadePool.ts
9102d0756f7af505925907d0660efa71371f75a0cb1f82d7fe7b4c5230e57b2c ./test/TestSettle.ts
6fea96ea544c39e33d849c55130687d2b4bd900abb3978d0c4ea9ef2067eef56 ./test/TestEncoding.ts
```

Changelog

- 2022-08-02 - Initial report
- 2023-03-03 - Fix review update
- 2023-03-14 - Second fix review update

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Aave, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.