Quantstamp

Alchemy - LightAccount V2

Executive Summary

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

Type	ERC-4337 Account
Timeline	2024-03-22 through 2024-03-28
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	ERC-4337 ERC-7562
Source Code	• alchemyplatform/light-account #93f46a2
Auditors	• Nikita Belenkov Auditing Engineer • Shih-Hung Wang Auditing Engineer • Gereon Mendler Auditing Engineer • Ruben Koch Senior Auditing Engineer

Documentation quality	High
Test quality	High
Total Findings	4 Fixed: 3 Mitigated: 1
High severity findings	0
Medium severity findings	0
Low severity findings	1 Fixed: 1
Undetermined severity findings	0
Informational findings	3 Fixed: 2 Mitigated: 1

Summary of Findings

In this audit, we reviewed the second version of the `LightAccount` developed by the Alchemy team. The main changes that this version introduced included updating implementation to comply with the ERC-4337 v0.7 and the new rules in ERC-7562, the introduction of a `LightAccount` with multiple owners called `MultiOwnerLightAccount`, and other smaller improvements.

Overall, the code is well-written and follows very good software development practices. We have found minor issues ranging from assembly not clearing upper bits to signature verification not fully following the ERC-4337 specification. These few small issues should all have straightforward fixes and should be addressed before deployment.

Some issues from the original audit of version 1 also apply but have not been included in this report, namely ALC-1 and ALC-2, which outline more general concerns around multiple user operations getting rejected. ALC-3 also applies, as adding expiry to the EOA signature is a good practice. ALC-4 should also followed, so that the one-step ownership transfer is documented for `v2` and also for the newly created `MultiOwnerLightAccount`.

The test suite consists of 118 tests, of which all pass successfully. The branch coverage stands a decent 86.52%, which could still be slightly improved.

Fix Review

All issues have been either fixed or mitigated by the Alchemy team in the commit `0a9480081131c58843a759301b967b9eac99816e`. The test suite has been adequately updated to accommodate the changes.

ID	DESCRIPTION	SEVERITY	STATUS
ALC-LA2-1	**`SIG_VALIDATION_FAILED` Shouldn't Be Returned in All the Cases of ECDSA Failure**	• Low	Fixed
ALC-LA2-2	**Uncleared Upper Bits of Variables in Inline Assembly**	• Informational	Mitigated
ALC-LA2-3	**Potential Gas Griefing by Changing the Signature Verification**	• Informational	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
	Mode		
ALC-LA2-4	Missing Input Validation	● Informational	Fixed

Assessment Breakdown

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

Disclaimer
Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

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

1. Code review that includes the following
   1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   3. 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:
   1. 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.
   2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

The following files were in scope:
1. `src/common/BaseLightAccount.sol`
2. `src/common/BaseLightAccountFactory.sol`
3. `src/common/ERC1271.sol`
4. `src/common/CustomSlotInitializable.sol`
5. `src/LightAccount.sol`
6. `src/LightAccountFactory.sol`
7. `src/MultiOwnerLightAccount.sol`
8. `src/MultiOwnerLightAccountFactory.sol`

Findings

ALC-LA2-1
`SIG_VALIDATION_FAILED` Shouldn't Be Returned in All the Cases of ECDSA Failure

● Low ⓘ Fixed

Update
`digest.recover()` is now used instead of `digest.tryRecover()`. This means that the function would revert when a malformed signature is used. Fixed in commit `3648cb0cdeb420ad54d6c04d5964d04d05698db1`

File(s) affected: `src/MultiOwnerLightAccount.sol`, `src/LightAccount.sol`

Description: When an EOA signature is submitted, it is decoded via `digest.tryRecover(signature)`. This function call returns the address recovered and an error of type `ECDSA.RecoverError`. There are currently 4 entries in the `ECDSA.RecoverError` enum:

```
1. NoError,
2. InvalidSignature,
3. InvalidSignatureLength,
4. InvalidSignatureS
```

In the current iteration of the code base, in the case of the 3 invalid errors, `SIG_VALIDATION_FAILED` is returned, which is not entirely correct, as it should only be returned if a signature is correctly formed but the validation fails. The only case where `SIG_VALIDATION_FAILED` should be returned is `NoError`, but the signer does not match the expected `owner`, which means that the signature is correctly formed, but the validation fails. In the case of `InvalidSignatureLength`, `InvalidSignatureS` and `InvalidSignature`, the signature itself is invalid; in the first case, the signature is not of length `65`. In the second error, the `s` value is in the upper range, and in the last error, either `r = 0`, `s = 0`, or `r` is too large.

Recommendation: In the 3 error cases, i.e., `InvalidSignatureLength`, `InvalidSignatureS` and `InvalidSignature`, the call should revert instead of `SIG_VALIDATION_FAILED` being returned. This can be achieved by using `digest.recover()` instead of `digest.tryRecover()`.

ALC-LA2-2
Uncleared Upper Bits of Variables in Inline Assembly

● Informational ⓘ Mitigated

Update
Marked as "Mitigated" by the client.
Code-comment expecting the caller to have cleared the upper bits has been added in: `805fb8fbc37a3b44c4b051ff0c672601f82020f2`.
The client provided the following explanation:

> Because the owner parameter is passed in from an external call parameter, the upper bits should already be clean.

File(s) affected: `src/LightAccountFactory.sol`

Description: In the `_getCombinedSalt()` function of `LightAccountFactory`, the input parameter `owner` is an address, and hence the upper bits are not guaranteed to be 0 when reading its value in assembly. According to the latest Solidity Inline Assembly documentation at the time of writing:

> "If you access variables of a type that spans less than 256 bits (for example uint64, address, or bytes16), you cannot make any assumptions about bits not part of the encoding of the type. Especially, do not assume them to be zero. To be safe, always clear the data properly before you use it in a context where this is important [...]"

Any dirty upper bit in `owner` may lead to a different result of `_getCombinedSalt()` and, therefore, a different account address.

Recommendation: Consider clearing the upper bits of the `owner` parameter in the assembly block before using it.

ALC-LA2-3
Potential Gas Griefing by Changing the Signature Verification Mode

● Informational ⓘ Fixed

Update

`SignatureType.CONTRACT` has been removed from `MultiOwnerLightAccount`. Fixed in commit `524ee491d48bbecaa113c21ae2e2c4f5b6ba47d3`.

File(s) affected: `src/MultiOwnerLightAccount.sol`

Description: The `_validateSignature()` function of `MultiOwnerLightAccount` allows the user operation to be validated in three different modes based on the first byte of `userOp.signature`. The mode `SignatureType.CONTRACT` iterates through the owner list until it finds an owner contract accepting the given signature. On the other hand, the mode `SignatureType.CONTRACT_WITH_ADDR` only checks if one owner contract accepts the signature, which provides potential gas savings as the iteration through the owner list is avoided.

However, an adversary who listens to the user operation requests, such as a bundler, can modify the first byte of `userOp.signature` and force the account to use a more gas-expensive signature validation mode by switching from `CONTRACT_WITH_ADDR` to `CONTRACT`. If the specified verification gas limit in the user operation is sufficiently high, the signature validation can still pass since the only-provided signature is correct.

Note that the user operation signer is usually not incentivized to set an overly high verification gas limit. Also, the incentive for an adversary to perform such an attack seems low since they do not directly gain profit. Due to the above reasons, this issue is considered Info severity.

Recommendation: Consider documenting this case in public-facing documentation and recommending that account owners not set unnecessarily high verification gas limits for user operations. Also, consider storing the values of the signature type and the possible owner address as part of the signed values of the UserOperation struct. Alternatively, remove the `SignatureType.CONTRACT` branch from the UserOperation validation via `_validateSignature()`.

ALC-LA2-4 Missing Input Validation

● Informational Fixed

Update
Relevant checks have been added as recommended. Fixed in commit `90af2aea986db732d132a83994db584116ffaecd`.

File(s) affected: `src/common/BaseLightAccount.sol`, `src/common/BaseLightAccountFactory.sol`

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. Specifically, in the following functions:

1. `BaseLightAccount.withdrawDepositTo()` should check that `withdrawAddress` is not `address(0)`.
2. `BaseLightAccountFactory.withdrawStake()` should check that `to` is not `address(0)`.
3. `BaseLightAccountFactory.withdraw()` should check that `to` is not `address(0)`.

Recommendation: We recommend adding the relevant checks.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – 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.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **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).

Adherence to Best Practices

1. Acknowledged In `MultiOwnerLightAccount._initialize()`, the event should be emitted after the call to `_updateOwners()` for code consistency.

2. Acknowledged Consider adjusting the `OwnersUpdated` event emitted in `MultiOwnerLightAccount._updateOwners()` to also emit the `msg.sender`, so the event log also contains the authorization used.
3. Fixed The check for `ownerToAdd == address(0)` in `MultiOwnerLightAccount._addOwnersOrRevert()` is already taken care of by the call to `LinkedListSetLib.tryAdd()` and can therefore be removed from the conditional statement.
4. Fixed In `MultiOwnerLightAccountFactory.createAccountSingle()`, a check that `owner != address(0)` could replace a call to `_validateOwnersArray()` and save some gas.
5. Fixed With ERC-4337 v0.7, support for the ERC-165 interface compliance check has been added to the `EntryPoint`. It is a good practice to verify the address is compliant with the expected interface before adding it to the contract.
6. Fixed In `BaseLightAccount`, the `SIG_VALIDATION_FAILED` constant is currently imported twice from `account-abstraction/core/Helpers.sol`.
7. Fixed Light account uses custom slots in memory for storage and initialization values, so that the storage of the contract remains upgradable. The current iteration of the contract is `v2`, but the namespaces have been left at `v1` in these cases. This could lead to unwanted storage collisions for uninformed users. Consider adding an explanatory in-line comment or updating to the `v2` version.
8. Fixed The following functions can be marked as `external`:
    - BaseLightAccount
        - `addDeposit()`
        - `withdrawDepositTo()`
        - `getDeposit()`
    - LightAccount
        - `initialize()`
    - LightAccountFactory
        - `createAccount()`
        - `getAddress()`
    - MultiOwnerLightAccountFactory
        - `createAccount()`
        - `createAccountSingle()`
        - `getAddress()`

Toolset

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

Setup

Tool Setup:

- Slither v0.10.0

Steps taken to run the tools:

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

Automated Analysis

Slither

Slither found 644 potential issues. Most of them were found in the test files and marked as false positives. All of them were discussed in this report or classified as false positives.

Test Suite Results

The test suite consists of 118 tests, of which all pass. It is recommended to add bundler integration tests to improve confidence in ERC-4337 compliance.

Fix Review

The test suite has been extended to 128 tests, of which all pass. The tests have been added to cover the changes made during the audit.

```
Running 7 tests for test/CustomSlotInitializable.t.sol:CustomSlotInitializableTest
[PASS] testCannotCallDisableInitializersInInitializer() (gas: 183809)
[PASS] testCannotReinitialize() (gas: 21393)
[PASS] testCannotUpgradeBackwards() (gas: 37656)
[PASS] testDisableInitializers() (gas: 31642)
[PASS] testIsInitializing() (gas: 227011)
[PASS] testSimpleInitialization() (gas: 569719)
[PASS] testUpgrade() (gas: 32907)
Test result: ok. 7 passed; 0 failed; 0 skipped; finished in 22.70ms
```

```
Running 11 tests for test/LightAccountFactory.t.sol:LightAccountFactoryTest
[PASS] test2StepOwnershipTransfer() (gas: 30398)
[PASS] testAddStake() (gas: 59850)
[PASS] testCannotRenounceOwnership() (gas: 10772)
[PASS] testGetAddress() (gas: 129472)
[PASS] testReturnsAddressWhenAccountAlreadyExists() (gas: 129209)
[PASS] testRevertWithInvalidEntryPoint() (gas: 67864)
[PASS] testUnlockStake() (gas: 65460)
[PASS] testWithdraw() (gas: 60374)
[PASS] testWithdrawStake() (gas: 60457)
[PASS] testWithdrawStakeToZeroAddress() (gas: 67131)
[PASS] testWithdrawToZeroAddress() (gas: 55591)
Test result: ok. 11 passed; 0 failed; 0 skipped; finished in 979.04ms

Running 18 tests for test/MultiOwnerLightAccountFactory.t.sol:MultiOwnerLightAccountFactoryTest
[PASS] test2StepOwnershipTransfer() (gas: 30424)
[PASS] testAddStake() (gas: 59949)
[PASS] testCannotRenounceOwnership() (gas: 10849)
[PASS] testGetAddress() (gas: 162764)
[PASS] testGetAddressAndCreateAccountWithDescendingOwners() (gas: 50357)
[PASS] testGetAddressAndCreateAccountWithDuplicateOwners() (gas: 50408)
[PASS] testGetAddressAndCreateAccountWithEmptyOwners() (gas: 12968)
[PASS] testGetAddressAndCreateAccountWithMaxOwners() (gas: 4941561)
[PASS] testGetAddressAndCreateAccountWithTooManyOwners() (gas: 2365636)
[PASS] testGetAddressAndCreateAccountWithZeroAddress() (gas: 46406)
[PASS] testGetAddressSingle() (gas: 160973)
[PASS] testReturnSameAddressWhenAccountAlreadyExists() (gas: 163861)
[PASS] testRevertWithInvalidEntryPoint() (gas: 68672)
[PASS] testUnlockStake() (gas: 65493)
[PASS] testWithdraw() (gas: 60512)
[PASS] testWithdrawStake() (gas: 60629)
[PASS] testWithdrawStakeToZeroAddress() (gas: 67274)
[PASS] testWithdrawToZeroAddress() (gas: 55712)
Test result: ok. 18 passed; 0 failed; 0 skipped; finished in 979.53ms

Running 41 tests for test/LightAccount.t.sol:LightAccountTest
[PASS] testAddDeposit() (gas: 61252)
[PASS] testCannotInitializeWithZeroOwner() (gas: 2481606)
[PASS] testCannotTransferOwnershipToCurrentOwner() (gas: 19700)
[PASS] testCannotTransferOwnershipToLightContractItself() (gas: 20123)
[PASS] testCannotTransferOwnershipToZero() (gas: 19684)
[PASS] testEntryPointCanTransferOwnership() (gas: 148582)
[PASS] testEntryPointCanUpgrade() (gas: 1610402)
[PASS] testEntryPointGetter() (gas: 13095)
[PASS] testExecuteBatchCalledByOwner() (gas: 48548)
[PASS] testExecuteBatchFailsForUnevenInputArrays() (gas: 22036)
[PASS] testExecuteBatchWithValueCalledByOwner() (gas: 56241)
[PASS] testExecuteBatchWithValueFailsForUnevenInputArrays() (gas: 22723)
[PASS] testExecuteCanBeCalledByEntryPointWithExternalOwner() (gas: 168369)
[PASS] testExecuteCanBeCalledByOwner() (gas: 47739)
[PASS] testExecuteCannotBeCalledByRandos() (gas: 18922)
[PASS] testExecuteRevertingCallShouldRevertWithSameData() (gas: 98843)
[PASS] testExecuteWithValueCanBeCalledByOwner() (gas: 53900)
[PASS] testExecutedCanBeCalledByEntryPointWithContractOwner() (gas: 179719)
[PASS] testFuzz_rejectsUserOpsWithInvalidSignatureType(uint8) (runs: 5000, μ: 42702, ~: 42689)
[PASS] testInitialize() (gas: 2516388)
[PASS] testIsValidSignatureForContractOwner() (gas: 41365)
[PASS] testIsValidSignatureForEoaOwner() (gas: 28771)
[PASS] testIsValidSignaturePersonalSign() (gas: 27417)
[PASS] testIsValidSignaturePersonalSignForContractOwner() (gas: 39070)
[PASS] testIsValidSignaturePersonalSignRejectsInvalid() (gas: 39711)
[PASS] testIsValidSignatureRejectsInvalid() (gas: 47475)
[PASS] testNonOwnerCannotUpgrade() (gas: 1433046)
[PASS] testOwnerCanTransferOwnership() (gas: 26240)
[PASS] testOwnerCanUpgrade() (gas: 1490894)
[PASS] testRandosCannotTransferOwnership() (gas: 17421)
[PASS] testRejectsUserOpsWithInvalidSignature() (gas: 81995)
[PASS] testRevertsUserOpsWithMalformedSignature() (gas: 96241)
[PASS] testSelfCanTransferOwnership() (gas: 151026)
[PASS] testSelfCanUpgrade() (gas: 1613819)
[PASS] testStorageSlots() (gas: 9743)
[PASS] testValidateInitCodeHash() (gas: 34504)
```

```
[PASS] testWithdrawDepositCanBeCalledByEntryPointWithExternalOwner() (gas: 184048)
[PASS] testWithdrawDepositCanBeCalledBySelf() (gas: 188109)
[PASS] testWithdrawDepositToCalledByOwner() (gas: 100476)
[PASS] testWithdrawDepositToCannotBeCalledByRandos() (gas: 58547)
[PASS] testWithdrawDepositToZeroAddress() (gas: 60237)
Test result: ok. 41 passed; 0 failed; 0 skipped; finished in 979.71ms

Running 51 tests for test/MultiOwnerLightAccount.t.sol:MultiOwnerLightAccountTest
[PASS] testAddAndRemoveSameOwner() (gas: 30568)
[PASS] testAddDeposit() (gas: 61667)
[PASS] testCannotAddExistingOwner() (gas: 20918)
[PASS] testCannotAddLightContractItselfAsOwner() (gas: 20748)
[PASS] testCannotAddZeroAddressAsOwner() (gas: 20335)
[PASS] testCannotInitializeWithZeroOwner() (gas: 2985962)
[PASS] testCannotRemoveAllOwners() (gas: 28666)
[PASS] testEntryPointCanUpdateOwners() (gas: 179321)
[PASS] testEntryPointCanUpgrade() (gas: 1610553)
[PASS] testEntryPointGetter() (gas: 13292)
[PASS] testExecuteBatchCalledByOwner() (gas: 48753)
[PASS] testExecuteBatchFailsForUnevenInputArrays() (gas: 22130)
[PASS] testExecuteBatchWithValueCalledByOwner() (gas: 56471)
[PASS] testExecuteBatchWithValueFailsForUnevenInputArrays() (gas: 22845)
[PASS] testExecuteCanBeCalledByEntryPointWithContractOwnerSpecified() (gas: 207355)
[PASS] testExecuteCanBeCalledByEntryPointWithExternalOwner() (gas: 168744)
[PASS] testExecuteCanBeCalledByOwner() (gas: 48131)
[PASS] testExecuteCannotBeCalledByRandos() (gas: 19039)
[PASS] testExecuteRevertingCallShouldRevertWithSameData() (gas: 98982)
[PASS] testExecuteWithValueCanBeCalledByOwner() (gas: 54083)
[PASS] testFuzz_isValidSignatureRejectsInvalidSignatureType(uint8) (runs: 5000, μ: 27014, ~: 26998)
[PASS] testFuzz_rejectsUserOpsWithInvalidSignatureType(uint8) (runs: 5000, μ: 42817, ~: 42801)
[PASS] testInitialize() (gas: 3140108)
[PASS] testIsValidSignatureForContractOwnerSpecified() (gas: 68091)
[PASS] testIsValidSignatureForEoaOwner() (gas: 29130)
[PASS] testIsValidSignaturePersonalSign() (gas: 27815)
[PASS] testIsValidSignaturePersonalSignForContractOwnerSpecified() (gas: 66588)
[PASS] testIsValidSignaturePersonalSignRejectsContractOwnerUnspecified() (gas: 64958)
[PASS] testIsValidSignaturePersonalSignRejectsInvalid() (gas: 40254)
[PASS] testIsValidSignatureRejectsContractOwnerUnspecified() (gas: 67098)
[PASS] testIsValidSignatureRejectsInvalidContractOwner() (gas: 29964)
[PASS] testIsValidSignatureRejectsInvalidEOA() (gas: 47777)
[PASS] testNonOwnerCannotUpgrade() (gas: 1433404)
[PASS] testOwnerCanUpdateOwners() (gas: 57267)
[PASS] testOwnerCanUpgrade() (gas: 1491252)
[PASS] testRandosCannotUpdateOwners() (gas: 18418)
[PASS] testRejectsUserOpWithContractOwnerUnspecified() (gas: 86812)
[PASS] testRejectsUserOpWithInvalidContractOwnerSpecified() (gas: 90441)
[PASS] testRejectsUserOpWithPartialContractOwnerSpecified() (gas: 85033)
[PASS] testRejectsUserOpsWithInvalidSignature() (gas: 82344)
[PASS] testRemoveNonexistantOwner() (gas: 23159)
[PASS] testRevertsUserOpsWithMalformedSignature() (gas: 96476)
[PASS] testSelfCanUpdateOwners() (gas: 183505)
[PASS] testSelfCanUpgrade() (gas: 1614004)
[PASS] testStorageSlots() (gas: 9885)
[PASS] testValidateInitCodeHash() (gas: 42947)
[PASS] testWithdrawDepositCanBeCalledByEntryPointWithExternalOwner() (gas: 184225)
[PASS] testWithdrawDepositCanBeCalledBySelf() (gas: 188518)
[PASS] testWithdrawDepositToCalledByOwner() (gas: 100859)
[PASS] testWithdrawDepositToCannotBeCalledByRandos() (gas: 58930)
[PASS] testWithdrawDepositToZeroAddress() (gas: 60422)
Test result: ok. 51 passed; 0 failed; 0 skipped; finished in 3.51s

Ran 5 test suites: 128 tests passed, 0 failed, 0 skipped (128 total tests)
```

Code Coverage

The branch coverage is 86.52%, which could be improved to recommended 90%+.

Fix Review

The branch coverage has been improved and now stands at 87.36%.

File	% Lines	% Statements	% Branches	% Funcs
src/LightAccount.sol	97.78% (**44/**45)	98.53% (**67/**68)	94.44% (**17/**18)	100.00% (**12/**12)
src/LightAccountFactory.sol	100.00% (**7/**7)	100.00% (**8/**8)	100.00% (**2/**2)	100.00% (**3/**3)
src/MultiOwnerLightAccount.sol	100.00% (**53/**53)	100.00% (**81/**81)	94.44% (**17/**18)	100.00% (**14/**14)
src/MultiOwnerLightAccountFactory.sol	100.00% (**28/**28)	100.00% (**34/**34)	83.33% (**10/**12)	100.00% (**5/**5)
src/common/BaseLightAccount.sol	100.00% (**22/**22)	100.00% (**38/**38)	100.00% (**10/**10)	100.00% (**11/**11)
src/common/BaseLightAccountFactory.sol	73.33% (**11/**15)	78.95% (**15/**19)	60.00% (**6/**10)	83.33% (**5/**6)
src/common/CustomSlotInitializable.sol	83.33% (**10/**12)	83.33% (**10/**12)	66.67% (**4/**6)	80.00% (**4/**5)
src/common/ERC1271.sol	100.00% (**2/**2)	100.00% (**3/**3)	100.00% (**0/**0)	100.00% (**1/**1)

Changelog

- 2024-03-28 - Initial report
- 2024-04-09 - Final report

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, 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

Notice of confidentiality

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.

Quantstamp