Security Assessment & Formal Verification Final Report

Lido Dual Governance

February 2025

Prepared for Lido

certora

LIDO

Table of contents

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform

Lido Dual Governance	https://github.com/lidofinance/dual-governance	Start: 46d667e End : 3e0f1ae	EVM

Note

This document describes Certora's second audit and formal verification of Lido Dual Governance after the Dual Governance source code underwent revisions. Details about our earlier audit and verification on the earlier version of Dual Governance can be found in Certora's earlier report.

Project Overview

This document describes the specification and verification of **Lido Dual Governance** using the Certora Prover and manual code review findings. The work was undertaken from **January 10 2025** to **February 7 2025**

The following contract list is included in our scope:

- contracts/Escrow.sol
- contracts/libraries/AssetsAccounting.sol
- contracts/DualGovernance.sol
- contracts/libraries/DualGovernanceStateMachine.sol
- contracts/EmergencyProtectedTimelock.sol
- contracts/libraries/WithdrawalBatchesQueue.sol
- contracts/committees/HashConsensus.sol
- contracts/libraries/ExecutableProposals.sol
- contracts/libraries/Tiebreaker.sol
- contracts/libraries/EmergencyProtection.sol
- contracts/libraries/DualGovernanceConfig.sol
- contracts/libraries/EscrowState.sol
- contracts/libraries/Proposers.sol
- contracts/libraries/DualGovernanceStateTransitions.sol
- contracts/types/Duration.sol
- contracts/committees/TiebreakerCore.sol
- contracts/committees/TiebreakerSubCommittee.sol
- contracts/libraries/EnumerableProposals.sol
- contracts/ImmutableDualGovernanceConfigProvider.sol
- contracts/types/Timestamp.sol
- contracts/libraries/TimelockState.sol
- contracts/types/ETHValue.sol

- contracts/libraries/SealableCalls.sol
- contracts/ResealManager.sol
- contracts/types/SharesValue.sol
- contracts/TimelockedGovernance.sol
- contracts/types/PercentD16.sol
- contracts/types/IndexOneBased.sol
- contracts/committees/ProposalsList.sol
- contracts/libraries/ExternalCalls.sol
- contracts/Executor.sol
- contracts/utils/arrays.sol

The Certora Prover demonstrated that the implementation of the **Solidity** contracts above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of all the Solidity contracts listed above. During the verification process and the manual audit, the Certora team discovered bugs in the Solidity contracts code, as listed on the following page.

Protocol Overview

Currently, the Lido protocol governance consists of the Lido DAO that uses LDO voting to approve DAO proposals, along with an optimistic voting subsystem called Easy Tracks that is used for routine changes of low-impact parameters and falls back to LDO voting given any objection from LDO holders.

Additionally, there is a Gate Seal emergency committee that allows pausing certain protocol functionality (e.g. withdrawals) for a pre-configured amount of time sufficient for the DAO to vote on and execute a proposal. The Gate Seal committee can only enact a pause once before losing its power (so it has to be re-elected by the DAO after that).

The Dual governance mechanism (DG) is an iteration on the protocol governance that gives stakers a say by allowing them to block DAO decisions and providing a negotiation device between stakers and the DAO.

Another way of looking at dual governance is that it implements 1) a dynamic user-extensible timelock on DAO decisions and 2) a rage quit mechanism for stakers taking into account the specifics of how Ethereum withdrawals work.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	0	0	0
High	0	0	0
Medium	4	4	3
Low	2	2	1
Total	6	6	4

Severity Matrix

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

certora

LIDO

Detailed Findings

ID	Title	Severity	Status
M-01	sanity checks missing	Medium	Partially Fixed
M-02	No check that the new adminExecutor is an executor	Medium	Acknowledged
M-03	Possible overflow than can cause the DG to get stuck in the vetoSignalling state	Medium	Fixed
M-04	It is possible to execute without waiting the MIN_EXECUTION_DELAY	Medium	Fixed
L-01	VetoSignal the escrow for twice the time user should be able to	Low	Acknowledged
L-02	Tiebreaker can vote to resume a contract even when it is not paused	Low	Fixed

Medium Severity Issues

M-01 - sanity checks missing

Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: DualGovernanceConfig.sol, Escrow.sol	Status: Partially Fixed	

Description:

Should add sanity checks that the parameters are in the valid range.
`secondSealRageQuitSupport` should not be more than 100%
`firstSealRageQuitSupport` should not be 0%
`vetoSignallingMinDuration` should not be 0
`rageQuitEthWithdrawalsMaxDelay` should not be max_uint32
Also in the escrow in the `initialize`, `minAssetsLockDuration` is not checked etc

Recommendation:
Add checks accordingly.

Lido's response: Fixed in PR-257

M-02 - No check that the new adminExecutor is an executor

Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: EmergencyProtectedTimelock.sol	Status: Acknowledged	

Description: if the admin executor set a non executor to the adminExecutor the DG cannot work. This can be very bad, the whole system is stuck without an adminExcutor and the only recovery will be the emergency reset of the governance.

```
File: EmergencyProtectedTimelock.sol

     function setAdminExecutor(address newAdminExecutor) external {
        _timelockState.checkCallerIsAdminExecutor();
        _timelockState.setAdminExecutor(newAdminExecutor);
    }
```

Recommendation:
Add a 2 step transfer of the adminExecutor role and in that 2 step add a check that the new adminExecutor is indeed an executor.

Lido's response: The update of the admin executor is performed by the DAO through a Dual Governance proposal, and the probability of misconfiguration is considered very low. To eliminate this risk completely, any proposal updating the admin executor must include a final action that validates the new admin executor is properly registered in Dual Governance and reverts execution if the validation fails.

M-03 - Possible overflow than can cause the DG to get stuck in the vetoSignalling state

Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: DualGovernanceConfig.sol	Status: Fixed	

Description:

if there is an overflow for some configuration of `rageQuitEthWithdrawalsDelayGrowth` for example `max_uint32`, with `rageQuitRound * rageQuitEthWithdrawalsDelayGrowth` can overflow in the `plusSeconds` function which will DOS the DG and the escrow when a RageQuit should happen and the only way out is the Tiebreaker committee after the tiebreakerTimeout will pass.

```
File: DualGovernanceConfig.sol #Old

 function calcRageQuitWithdrawalsDelay(
        Context memory self,
        uint256 rageQuitRound
    ) internal pure returns (Duration) {
-         return Durations.min(
-             self.rageQuitEthWithdrawalsMinDelay.plusSeconds(
-                 rageQuitRound *
-self.rageQuitEthWithdrawalsDelayGrowth.toSeconds()
-             ),
-             self.rageQuitEthWithdrawalsMaxDelay
-         );
    }



File: DualGovernanceConfig.sol #New

 function calcRageQuitWithdrawalsDelay(
        Context memory self,
        uint256 rageQuitRound
    ) internal pure returns (Duration) {
```

```
+        uint256 rageQuitWithdrawalsDelayInSeconds =
+self.rageQuitEthWithdrawalsMinDelay.toSeconds()
+            + rageQuitRound *
+self.rageQuitEthWithdrawalsDelayGrowth.toSeconds();
+
+        return rageQuitWithdrawalsDelayInSeconds >
+self.rageQuitEthWithdrawalsMaxDelay.toSeconds()
+            ? self.rageQuitEthWithdrawalsMaxDelay
+            : Durations.from(rageQuitWithdrawalsDelayInSeconds);
    }
```

Recommendation:
Rewrite this in a way where even if this overflow you will get the intended behavior, like not using the `plusSeconds` function or have some if for a case of overflow and just take the other option.

Lido's response: Fixed in PR-257

M-04 - It is possible to execute without waiting the `MIN_EXECUTION_DELAY`

Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: ExecutableProposals.sol	Status: Fixed	

Description: If after submit delay is 0 and after schedule delay is non-zero in the case that we want to make the submit delay longer and the schedule delay 0 we will get a short time in which we can submit and execute the delay changes proposal, then we can execute immediately.

```
File: ExecutableProposals.sol #Old

    function execute(Context storage self, uint256 proposalId, Duration
afterScheduleDelay) internal {
        Proposal memory proposal = self.proposals[proposalId];

        _checkProposalNotCancelled(self, proposalId, proposal.data);

        if (proposal.data.status != Status.Scheduled) {
            revert UnexpectedProposalStatus(proposalId, proposal.data.status);
        }

        if (afterScheduleDelay.addTo(proposal.data.scheduledAt) >
Timestamps.now()) {
            revert AfterScheduleDelayNotPassed(proposalId);
        }

        self.proposals[proposalId].data.status = Status.Executed;

        ExternalCalls.execute(IExternalExecutor(proposal.data.executor),
proposal.calls);
        emit ProposalExecuted(proposalId);
    }

File: ExecutableProposals.sol #New

    function execute(
        Context storage self,
        uint256 proposalId,
        Duration afterScheduleDelay,
        Duration minExecutionDelay
```

```
    ) internal {
        Proposal memory proposal = self.proposals[proposalId];

        _checkProposalNotCancelled(self, proposalId, proposal.data);

        if (proposal.data.status != Status.Scheduled) {
            revert UnexpectedProposalStatus(proposalId, proposal.data.status);
        }

        if (afterScheduleDelay.addTo(proposal.data.scheduledAt) >
Timestamps.now()) {
            revert AfterScheduleDelayNotPassed(proposalId);
        }

+        if (minExecutionDelay.addTo(proposal.data.submittedAt) >
Timestamps.now()) {
+            revert MinExecutionDelayNotPassed(proposalId);
+        }

        self.proposals[proposalId].data.status = Status.Executed;

        ExternalCalls.execute(IExternalExecutor(proposal.data.executor),
proposal.calls);
        emit ProposalExecuted(proposalId);
    }
```

Recommendation:
Add a check in the execute that verifies that the time passed from the submit till the execute is at least the `MIN_EXECUTION_DELAY`

Lido's response: Fixed in PR-266

Low Severity Issues

L-01 - VetoSignal the escrow for twice the time user should be able to

Severity: **Low**	Impact: **Low**	Likelihood: **Medium**
Files: Escrow.sol	Status: Acknowledged	

Description:

Even with the minimum lock time we can use flashloans to reduce the total stEth needed to block the system (only by vetoSignalling not rageQuit) by half by locking the flashloan and unlocking from a different account.

Now we can lock `secondSealRageQuitSupport/2` of stEth and wait the minimum lock time after that pass. We can just flashloan `secondSealRageQuitSupport/2` of stEth and unlock the second half. This way we can make the system operate as if there is `secondSealRageQuitSupport` of stEth locked when there is only half of that locked.

Recommendation:
Maybe add a check that the total funds that are inside the escrow (for reactivating/continuing the VetoSignalling) are there for more than a single block, by doing so we avoid the flashloan tricks.

Lido's response: The impact of the described behavior remains strictly limited to delaying DAO proposals execution, as initiating a Rage Quit still requires locking the full `secondSealRageQuitSupport` amount in the signalling escrow. At the same time, the feasibility of this remains constrained by significant token amount requirements, including the initial capital needed to enter the `VetoSignalling` state and the ongoing flash loan fees for each attempt to prolong `VetoSignalling` beyond the `VetoSignallingDeactivation` duration.

L-02 - Tiebreaker can vote to resume a contract even when it is not paused

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: TiebreakerCoreCommittee.sol	Status: Fixed	

Description: if there is a vote that gets approved to resume an address but that address is not paused then the resume from the Tiebreaker will just wait until this address will be paused and then anyone can just unpause it immediately.

```
File: TiebreakerCoreCommittee.sol #Old

        function sealableResume(address sealable, uint256 nonce) external {
        _checkCallerIsMember();

        if (sealable == address(0)) {
            revert InvalidSealable(sealable);
        }

        if (nonce != _sealableResumeNonces[sealable]) {
            revert ResumeSealableNonceMismatch();
        }

        (bytes memory proposalData, bytes32 key) =
_encodeSealableResume(sealable, nonce);
        _vote(key, true);
        _pushProposal(key, uint256(ProposalType.ResumeSealable),
proposalData);
    }


File: TiebreakerCoreCommittee.sol #New

 function sealableResume(address sealable, uint256 nonce) external {
        _checkCallerIsMember();
+        checkSealableIsPaused(sealable);

        if (nonce != _sealableResumeNonces[sealable]) {
            revert ResumeSealableNonceMismatch();
        }
        (bytes memory proposalData, bytes32 key) =
```

```
_encodeSealableResume(sealable, nonce);
        _vote(key, true);
        _pushProposal(key, uint256(ProposalType.ResumeSealable),
proposalData);
    }
```

Recommendation:
Add a check that address is paused and also inside the blockers list

Lido's response: Fixed in PR-264

certora LIDO

Formal Verification

Verification Notations

Formally Verified	The rule is verified for every state of the contract(s), under the assumptions of the scope/requirements in the rule.
Formally Verified After Fix	The rule was violated due to an issue in the code and was successfully verified after fixing the issue
Violated	A counter-example exists that violates one of the assertions of the rule.

General Assumptions and Simplifications

For each of the contracts under verification, we rely on "mock" contracts that give artificial and simplified implementations of a few related contracts we do not have implementations of. We designed these to avoid any simplifications that overly limit the scope of verification. These are as follows:

- IStETH – we model this `DummyStETH.sol` as a simple ERC20 with a fixed exchange ratio of ETH * 5 / 3 = shares amount
- ERC20s – DummyERC20A / DummyERC20B implement relatively standard ERC20 contracts that are identical but allow the prover to choose different addresses for various ERC20 contracts
- DummyWstEth – implements a relatively standard ERC20 extended with wrap/unwrap functions
- IWithdrawalQueue – we implemented a simplified version of the real withdrawal queue that was designed to adequately capture the behavior of the real withdrawal queue
- We model the following functions as returning an arbitrary value on each invocation and assume they have no side-effects on the DualGovernance contract: Address.functionCallWithValue, ISealable.getResumeSinceTimestamp, IOwnable.transferOwnership, Executor.execute.

- Additionally, we assume functionCallWithValue, callGetResumeSinceTimestamp behave like math functions (i.e. it will return the same value on distinct invocations with the same parameters).

Formal Verification Properties

DualGovernance

Spec General Assumptions

- We assume the rage quit first seal threshold is greater than zero and the ragequit second seal is greater than the first seal

Spec Properties

P-01. Proposer indexes match their index in the array and are always < the array length	
Status: Formally Verified After Fix	*Assumption: we assume the proposer array is less than 5 to allow us to bound the iterations of loops.*

Rule Name	Status	Description	Link to rule report
w2_1a_indexes_match	Formally Verified after Fix	*for any registered proposer, his index should be ≤ the length of the array of proposers" and "for each entry in the struct in the array, show that the index inside is the same as the real array index* ***NOTE:*** *This originally caught a bug during which there was a counterexample. It now passes after Lido acknowledged and fixed the bug.* *Report with counterxample before bug fix.*	*Report*

Note: we ran this rule against both the code before the fix attempt was implemented and after the fix attempt was implemented (fix attempt commit link) This refers to finding C-01 from our earlier report.

P-02. Dual Governance Key Property 1

Status: Verified

Rule Name	Status	Description	Link to rule report
dg_kp_1_proposal_execution	Verified	Proposals cannot be executed in the Veto Signaling (both parent state and Deactivation sub-state) and Rage Quit states.	*Report*

Note: this property is meant to verify a rule from Lido's Key Properties documentation

P-03. Dual Governance Key Property 2

Status: Verified

Rule Name	Status	Description	Link to rule report
dg_kp_2_proposal_submission	Verified	*Proposals cannot be submitted in the Veto Signaling Deactivation sub-state or in the Veto Cooldown state.*	*Report*

P-04. Dual Governance Key Property 3

Status: Verified

Rule Name	Status	Description	Link to rule report
dg_kp_3_cooldown_execution	Verified	*If a proposal was submitted after the last time the Veto Signaling state was activated, then it cannot be executed in the Veto Cooldown state.*	*Report*

P-05. Dual Governance Key Property 4

Status: Verified

Rule Name	Status	Description	Link to rule report
dg_kp_4_single_ragequit	Verified	*One rage quit cannot start until the previous rage quit has been finalized. In other words, there can only be at most one active rage quit escrow at a time.*	*Report*

P-06. Dual Governance Key Property 4 Addendum

Status: Verified	Note: this only checks the state of the Veto Signaling Escrow after functions have completed and it does not check temporary changes part-way through function execution.

Rule Name	Status	Description	Link to rule report
dg_kp_4_single_ragequit_addendum	Verified	*The vetoSignalling Escrow is never in the RageQuit state.*	*Report*

P-07. Protocol Key Property 1

Status: Verified	

Rule Name	Status	Description	Link to rule report
pp_kp_1_ragequit_extends	Verified	*Regardless of the state in which a proposal is submitted, if the stakers are able to amass and maintain a certain amount of rage quit support before the ProposalExecutionMinTimelock expires, they can extend the timelock for a proportional time, according to the dynamic timelock calculation*	*Report*

P-08. Protocol Key Property 2

Status: Verified

Rule Name	Status	Description	Link to rule report
pp_kp_2_rageq uit_trigger	Verified	*PP-2: It's not possible to prevent a proposal from being executed indefinitely without triggering a rage quit.*	*Report*

P-09. Protocol Key Property 3

Status: Verified

Rule Name	Status	Description	Link to rule report
pp_kp_3_no_in definite_propos al_submission_ block	Verified	*PP-3: It's not possible to block proposal submission indefinitely.*	*Report*

P-10. Protocol Key Property 4

Status: Verified

Rule Name	Status	Description	Link to rule report
pp_kp_4_veto_signalling_deactivation_cancellable	Verified	*PP-4: Until the Veto Signaling Deactivation sub-state transitions to Veto Cooldown, there is always a possibility (given enough rage quit support) of canceling Deactivation and returning to the parent state (possibly triggering a rage quit immediately afterwards).*	*Report*

P-11. Proposal Submission States

Status: Verified

Rule Name	Status	Description	Link to rule report
dg_states_1_proposal_submission_states	Verified	*If proposal submission succeeds, the system was in one of these states: Normal, Veto Signalling, Rage Quit*	*Report*

P-12. Proposal Scheduling States

Status: Verified

Rule Name	Status	Description	Link to rule report
dg_states_2_proposal_scheduling_states	Verified	*If proposal scheduling succeeds, the system was in one of these states: Normal, Veto Cooldown*	*Report*

P-13. Only legal transitions are possible

Status: Verified

Rule Name	Status	Description	Link to rule report
dg_transitions_1_only_legal_transitions	Verified	*If proposal scheduling succeeds, the system was in one of these states: Normal, Veto Cooldown*	*Report*

P-14. Ragequit Round Resets in Veto Cooldown

Status: Verified

Rule Name	Status	Description	Link to rule report

ragequit_round_resets_in_veto cooldown	Verified	*Whenever the state transitions into VetoCooldown (with a change of state) the ragequit round becomes 0.*	*Report*

P-15. Cancel All Pending Proposals Caller

Status: Verified

Rule Name	Status	Description	Link to rule report
cancel_all_pending_proposals	Verified	*Calls to cancelAllPendingProposals will fail unless the caller is _proposalsCanceller.*	*Report*

P-16. Only legal transitions are possible

Status: Verified

Rule Name	Status	Description	Link to rule report
only_set_proposals_canceller_change_canceller	Verified	*No method other than setProposalsCanceller can change the address of _proposalsCanceller*	*Report*

Spec General Assumptions

Spec Properties

P-17. Executed is a terminal state for a proposal

Status: Formally Verified After Fix

Rule Name	Status	Description	Link to rule report
W1_4_TerminalityOfExecuted	Formally Verified After Fix	*Executed is a terminal state for a proposal, once executed it cannot transition to any other state* *NOTE: this was initially violated before a fix from Lido. Violated report prior to fix.* *Link to PR with fix*	*Report*

This refers to a finding from our earlier report.

P-18. Nonzero Proposals are within bounds

Status: Verified

Rule Name	Status	Description	Link to rule report
outOfBoundsProposalDoesNotExist	Verified	*Proposals with nonzero Ids must either have an ID in the range (0,proposalsCount] or have the NotExist status*	*Report*

P-19. Emergency Protected Timelock Key Property 1

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_KP_1_SubmissionToSchedulingDelay	Verified	*A proposal cannot be scheduled for execution before at least ProposalExecutionMinTimelock has passed since its submission.*	Report

P-20. Emergency Protected Timelock Key Property 2

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_KP_2_SchedulingToExecutionDelay	Verified	*A proposal cannot be executed until the emergency protection timelock has passed since it was scheduled.*	Report

P-21. Emergency Protection Configuration Guarded

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_1_EmergencyProtectionConfigurationGuarded	Verified	*Emergency protection configuration changes are guarded by committees or admin executors. We check here that the part of the state that should only be alterable by the respective emergency*	Report

		committees or through an admin proposal is indeed not changed on any method call other than ones correctly authorized .	

P-22. Only Governance Can Schedule

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_2a_Scheduling GovernanceOnly	Verified	*Only governance can schedule proposals.*	*Report*

P-23. Only Governance Can Submit Proposals

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_2b_Submission GovernanceOnly	Verified	*Only governance can submit proposals.*	*Report*

P-24. Emergency Mode Restriction

Status: Verified

Rule Name	Status	Description	Link to rule report

EPT_3_EmergencyModeExecutionRestriction	Verified	*If emergency mode is active, only emergency execution committee can execute proposals*	*Report*

P-25. Emergency Mode Liveness

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_9_EmergencyModeLiveness	Verified	*When emergency mode is active, the emergency execution committee can execute proposals successfully*	*Report*

P-26 .ProposalTimestampConsistency

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_10_ProposalTimestampConsistency	Verified	*Proposal timestamps reflect timelock actions*	*Report*

P-27. Terminality of Canceled

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_11_TerminalityOfCancelled	Verified	*Canceled is a terminal state for a proposal, once canceled it cannot transition to any other state*	*Report*

P-28. Governance changes cancels all proposals

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_12_GovChangeCancelsAll	Verified	*All proposals are canceled after a governance change. This is specified by showing that it is not possible to schedule any proposal after a call to setGovernance.*	*Report*

P-29. Combined delay is above the min execution delay

Status: Verified

Rule Name	Status	Description	Link to rule report

combined_delay_above_min_execution_delay	Verified	*The combined afterScheduleDelay and afterSubmitDelay is greater than or equal to MIN_EXECUTION_DELAY*	*Report*

P-30. Proposals with the SCHEDULED status must have been submitted getAfterSubmitDelay in the past

Status: Verified		Exception: This will not hold if setAfterSubmitDelay is called while there are scheduled proposals in-flight. However,

Rule Name	Status	Description	Link to rule report
scheduled_proposals_above_schedule_delay	Verified	*A proposal cannot have status Scheduled before at least getAfterSubmitDelay passes since the time it is submitted* *This invariant does not hold because it is possible for the delay to be changed by calling setAfterSubmitDelay while proposals are already in flight*	*Report*

P-31. Proposals with the EXECUTED status must have been submitted getAfterScheduleDelay in the past

Status: Verified	Exceptions: - This will not hold if a proposal is emergency executed (which is expected behavior). - This will not hold for a proposal that was already executed if setAfterScheduleDelay is called after execution. However executed proposals cannot be re-executed, so this is harmless.

Rule Name	Status	Description	Link to rule report

executed_proposals_above_schedule_delay	Verified	*A proposal cannot have status Executed before at least getAfterScheduleDelay passes since the time it is scheduled unless it is emergency executed.* *This invariant does not hold because it is possible for the delay to be changed by calling setAfterScheduleDelay while proposals are already in flight*	*Report*

P-32. Proposals must wait at least MIN_EXECUTION_TIME between submission and execution.

Status: Verified

Rule Name	Status	Description	Link to rule report
execute_waits_min_delay	Verified	*A proposal can only be successfully executed if at least MIN_EXECUTION_TIME has passed since it was submitted.*	*Report*

P-33. No submitted proposals have a submittedAt time in the future

Status: Verified

Rule Name	Status	Description	Link to rule report
noProposalsSubmittedInFuture	Verified	*All existing proposals (those with status other than NotExist) have a submittedAt time which is before or equal to the current timestamp*	*Report*

Escrow

Spec General Assumptions

- We assume the following function calls have no side effects on the Escrow contract and model these as returning arbitrary numbers (with no side effects). Essentially we assume these calls cannot re-enter the Escrow:
  - ResealManager: resume, reseal
    - Safety analysis: these are only callable by Governance. So this means we trust governance not to re-enter Escrow.
  - Timelock: submit, schedule, execute, cancelAllNonExecutedProposals, canSchedule, canExecute
    - Safety analysis: *canSchedule()* and *canExecute()* are view functions, so they may not reenter Escrow. *submit()*, *schedule()*, *cancelAllNonExecutedProposals()* are all callable only by Governance. It is technically possible for *execute()* to reenter the Escrow if the proposal targets Escrow, but this is guarded by the timelock and we expect stakeholders to reject such a proposal. Further, if the Escrow is already in use, it means the system is already in ragequit and it is not possible to execute in this case anyway.

Spec Properties

P-34. Batches Queue Close Front Running Resistance	
Status: Formally Verified After Fix	

Rule Name	Status	Description	Link to rule report
W2_2_front_running	Formally Verified After Fix	*In a situation where requestNextWithdrawalsBatch should close the queue, there is no way to prevent it from being closed by first calling another function. NOTE: This rule previously resulted in a counter-example when it was run against a bug in the Lido code: Counterexample Report*	*Report*

This refers to security finding H-06 from our earlier report

P-35. Batches Queue Close Final State

Status: Verified

Rule Name	Status	Description	Link to rule report
W2_2_batchesQueueCloseFinalState	Verified	*once requestNextWithdrawalsBatch results in batchesQueue.close() all additional calls result in close();*	*Report*

P-36. Escrow Key Property 1

Status: Verified

Rule Name	Status	Description	Link to rule report
E_KP_1_rageQuitSupportValue	Verified	*ignoring imprecisions due to fixed-point arithmetic, the rage quit support of an escrow is equal to the formula from the Lido Key Properties document*	*Report*

P-37. Escrow Key Property 3

Status: Verified

Rule Name	Status	Description	Link to rule report
E_KP_3_rageQuitNolockUnlock	Verified	*It's not possible to lock funds in or unlock funds from an escrow that is already in the rage quit state.*	*Report*

		locking/unlocking implies changing the stETHLockedShares or unstETHLockedShares of an account	

P-38. Escrow Key Property 4

Status: Verified

Rule Name	Status	Description	Link to rule report
E_KP_4_unlockMinTime	Verified	*An agent cannot unlock their funds until SignallingEscrowMinLockTime has passed since this user last locked funds.*	Report

P-39. Escrow Key Property 5

Status: Verified

Rule Name	Status	Description	Link to rule report
E_KP_5_rageQuitStarter	Verified	*only dual governance can start a rage quit*	Report

P-40. Escrow Rage Quit State Final

Status: Verified

Rule Name	Status	Description	Link to rule report
E_State_1_rageQuitFinalState	Verified	*If the state of an escrow is RageQuitEscrow, we can execute any method and it will still be in the same state afterwards. Essentially it is a terminal state.*	*Report*

P-41. Valid State Rules – Escrow Data Structures stay in a safe subset of their types.

Status: Verified

Rule Name	Status	Description	Link to rule report
validState_batchQueuesSum	Verified	*countofBatchIds is as expected*	*Report*
validState_batchesQueue_claimed_vs_actual_1	Verified	*If an id is within the claimed indexes then it is marked as claimed in the withdrawal queue*	*Report*
validState_batchesQueue_distinct_unstETHRecords	Verified	*All unstEth are less than the lastRequestId and first batch if exists.*	*Report*
validState_batchesQueue_monotonicity	Verified	*Valid state of withdrawalQueue:* *1. an id is claimed only if it a valid requestId and was finalized* *2. an id is finalized iff it is le lastFinalizedRequestId*	*Report*
validState_batchesQueue_ordering	Verified	*The first id in each entry is greater than the last in the previous entry*	*Report*

validState_batchesQueue_withdrawalQueue	Verified	*Validity of batch queue ids:* *1. The last id in the last entry is less than or equal to the lastRequestId in withdrawal queue* *2. Escrow is the owner of the listed ids*	*Report*
validState_claimedUnstEth	Verified	*Total claimed unstEth is the partial sum of claimed of the lastFinalizedRequestId+ 1*	*Report*
validState_nonInitialized	Verified	*Before initialization everything is zero*	*Report*
validState_partialSumMonotonicity_1	Verified	*partial sum of withdrawn is le partial sum of claimed, by at least the element that is claimed but not withdrawn.*	*Report*
validState_partialSumMonotonicity_2	Verified	*partial sum of two ids is as expected*	*Report*
validState_partialSumOfClaimedUnstETH	Verified	*claimed unstETHRecords properties:* *1. if an unstETHRecord is finalized (status 2) then it is marked as finalized and not claimed in the withdrawal queue* *2. if an unstETHRecord is claimed or withdrawn (status 3 or 4) then it is marked as finalized and claimed in the withdrawal queue*	*Report*
validState_ragequit	Verified	*Once rageQuit start, batch queues are either open or closed*	*Report*
validState_signalling validState_totalETHIds	Verified	*while in signaling no claims and no batch queues*	*Report*
validState_totalLockedShares	Verified	*Current sum of all locked shares is less or equal the total lockedShares*	*Report*

validState_withdrawalQueue	Verified	*Valid state of withdrawalQueue:* *1. an id is claimed only if it a valid requestId and was finalized* *2. an id is finalized iff it is le lastFinalizedRequestId*	*Report*
validState_withdrawnEth	Verified	*Total withdrawn unstEth is the partial sum of withdrawn of the lastFinalizedRequestId+ 1*	*Report*
valid_batchIndex	Verified	*Last claimed batch index is lt the length of batch queue, if exists*	*Report*

P-42. Escrow Key Property 2: Solvency

Status: Verified

Rule Name	Status	Description	Link to rule report
solvency_ETH	Verified	*The total valuation accounting for unstaked eth, claimed eth, locked staked eth, staked eth, locked shares, claimed unstaked eth, and withdrawn unstaked eth must be less than the native balance of the contract.*	*Report* *claimNext must be run separately to avoid timeouts:* *Report* *Report*
solvency_ETH_before_ragequit	Verified	*Before rage quit eth value of escrow can not be reduced*	*Report*
solvency_stETH_before_ragequit	Verified	*Total holding of stEth before rageQuit start is at least the value of lockedShared*	*Report*

solvency_zero WstEthBalance	Verified	*Total holding of wst_eth is zero as all wst_eth are converted to st_eth*	*Report*
solvency_batchesQueue_solvent_leftToClaim	Verified	*Those request id left to claim are indeed not claimed*	*Report*
solvency_batchesQueue_allClaimed	Verified	*When all NFTs are claimed (according to internal accounting), the last one has been claimed*	*Report*

Admin Executor Rule

Spec General Assumptions

- **startRageQuit, initialize, and setMinAssetsLockDuration are all modeled as returning a non-determinstic number and having no side effects on contracts under verification.**

Spec Properties

P-43. The AdminExecutor must be an executor

Status: Violated

Rule Name	Status	Description	Link to rule report
admin_executor_is_executor	Violated	*The AdminExecutor must always be an Executor. This is an invariant.* *This does not hold and the property is violated because setAdminExecutor can transfer the AdminExecutor role to an address that is not actually an Executor.* *This corresponds to security finding M-02*	*Report*

EPT Cancelling Rules

Spec General Assumptions

- Same as general verification assumptions

Spec Properties

P-44. Only governance can cancel

Status: Verified	

Rule Name	Status	Description	Link to rule report
EPT_C1_only_governance_can_cancel	Verified	*cancelAllNonExecutedProposals cannot be called by any address other than the governance address*	Report

P-45. Can't schedule after cancelling

Status: Verified	

Rule Name	Status	Description	Link to rule report
EPT_C2_cant_schedule_after_cancelling	Verified	*after cancelAllNonExecutedProposals is called, no previously submitted proposal can be scheduled at any point in time*	Report

P-46. Can't execute or emergency execute after cancelling

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_C3_cant_execute_or_emergencyexecute_after_cancelling	Verified	*after cancelAllNonExecutedProposals is called, no previously submitted proposal (including scheduled ones) can be executed or emergency executed at any point in time*	*Report*

EPT Emergency Activation Rules

Spec General Assumptions

- **This has all "Spec General Assumptions" from Emergency Protected Timelock**

Spec Properties

P-47. Emergency execute is not callable in normal mode

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_EA1_execute_not_in_normal	Verified	*emergencyExecute cannot be called in normal mode*	*Report*

P-48. Emergency mode may only be activated by the proper committee address

Status: Verified	

Rule Name	Status	Description	Link to rule report
EPT_EA2_activate_only_by_committee	Verified	*activateEmergencyMode cannot be called by any address other than the emergency activation committee address*	Report

P-49. activateEmergencyMode cannot be called in emergency mode

Status: Verified	

Rule Name	Status	Description	Link to rule report
EPT_EA3_activate_not_in_emergency_mode	Verified	*activateEmergencyMode cannot be called in emergency mode*	Report

P-50. activateEmergencyMode changes mode from normal mode to emergency mode

Status: Verified	Note: we prove this only for the case where emergencyModeDuration is greater than zero as emergency mode cannot be activated otherwise.

Rule Name	Status	Description	Link to rule report

EPT_EA4_activate_changes_to_emergency	Verified	*activateEmergencyMode changes mode from normal mode to emergency mode*	*Report*

P-51. activateEmergencyMode cannot be called after the end date

Status: Verified	We assume the block timestamp is less than 2^40 to avoid an overflow.

Rule Name	Status	Description	Link to rule report
EPT_EA5_activate_not_after_protection_end	Verified	*activateEmergencyMode cannot be called after emergency protection end date passes*	*Report*

P-52. Proposals cannot be emergency executed other than by the proper committee

Status: Verified	

Rule Name	Status	Description	Link to rule report
EPT_EA6_only_committee_can_emergency_execute	Verified	*a proposal cannot be emergency executed by any address other than the emergency execution committee address*	*Report*

P-53. A scheduled proposal can be emergency executed before the delay elapses

Status: Verified	

Rule Name	Status	Description	Link to rule report
EPT_EA7_can_emergency_execute_before_delay_passes	Verified	*a scheduled proposal can be emergency executed before the post-schedule delay passes*	*Report*

EPT Emergency Config Rules

Spec General Assumptions

- Same as general verification assumptions

Spec Properties

P-54. Only specific functions can enter or exit

Status: Verified	

Rule Name	Status	Description	Link to rule report
EPT_EC1_only_specific_functions_can_enter_or_exit	Verified	*emergency mode can only be entered or exited as a result of one of the following calls: activateEmergencyMode, deactivateEmergencyMode, emergencyReset*	*Report*

P-55. Emergency activation committee address change scoping

Status: Verified	

Rule Name	Status	Description	Link to rule report
EPT_EC2_only_specific_functions_can_change_activation_committee	Verified	*emergency activation committee address can only be changed as a result of one of the following calls: setEmergencyProtectionActivationCommittee, deactivateEmergencyMode, emergencyReset*	*Report*

P-56. Emergency execution committee address change scoping

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_EC3_only_specific_functions_can_change_execution_committee	Verified	*emergency execution committee address can only be changed as a result of one of the following calls: setEmergencyProtectionExecutionCommittee, deactivateEmergencyMode, e*	*Report*

P-57. Emergency governance address change scoping

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_EC4_only_specific_function_can_change_governance	Verified	*emergency governance address can only be changed as a result of a setEmergencyGovernance call*	*Report*

P-58. Emergency mode duration change scoping

Status: Verified

Rule Name	Status	Description	Link to rule report

EPT_EC5_only_specific_functions_can_change_duration	Verified	*emergency mode duration can only be changed as a result of one of the following calls: setEmergencyModeDuration, deactivateEmergencyMode, emergencyReset*	*Report*

P-59. Emergency protection end date change scoping

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_EC6_only_specific_functions_can_change_protection_end	Verified	*emergency_protection end date can only be changed as a result of one of the following calls: setEmergencyProtectionEndDate, deactivateEmergencyMode, emergencyReset*	*Report*

P-60. Admin Executor privilege required for several calls

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_EC7_only_admin_can_call	Verified	*setEmergencyProtectionActivationCommittee, setEmergencyProtectionExecutionCommittee, setEmergencyGovernance, setEmergencyModeDuration,*	*Report*

		setEmergencyProtectionEndDate cannot be called by any address other than the admin executor address	

EPT Emergency Deactivation Rules

Spec General Assumptions

- Same as general verification assumptions

Spec Properties

P-61 Deactivation and reset only possible in emergency mode

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_ED1_only_in_emergency_mode	Verified	*deactivateEmergencyMode and emergencyReset can only be called in emergency mode*	Report

P-62 Anyone can deactivate after timeout

Status: Verified

Rule Name	Status	Description	Link to rule report

EPT_ED2_anyone_can_deactivate_after_timeout	Verified	*deactivateEmergencyMode can be called by anyone if emergency mode max duration passed since emergency mode activation*	*Report*

P-63 Only admin can deactivate before timeout

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_ED3_only_admin_can_deactivate_before_timeout	Verified	*deactivateEmergencyMode cannot be called by any address other than the admin executor address if emergency mode max duration did not pass since emergency mode activation*	*Report*

P-64 Emergency reset caller

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_ED4_only_execution_committee_can_reset	Verified	*emergencyReset cannot be called by any address other than the emergency execution committee address*	*Report*

P-65 Deactivate and emergency reset both deactivate emergency mode

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_ED5_deactivate_and_reset_actually_deactivate	Verified	*deactivateEmergencyMode and emergencyReset deactivate emergency mode*	*Report*

P-66 Deactivate and reset both zero out the context

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_ED6_deactivate_and_reset_nullify_context	Verified	*deactivateEmergencyMode and emergencyReset set emergency activation committee address, emergency execution committee address, emergency mode duration, and emergency protection end date to zero*	*Report*

P-67 Reset changes governance address

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_ED7_reset_changes_governance_address	Verified	*emergencyReset changes governance address to the emergency governance address*	*Report*

P-68 No proposals can be executed after emergency mode deactivation

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_ED8_no_proposals_after_deactivate_or_reset	Verified	*after deactivateEmergencyMode or emergencyReset is called, no previously submitted proposal (including scheduled ones) can be executed or emergency executed at any point in time*	*Report*

EPT General Config State

Spec General Assumptions

- Same as general verification assumptions

Spec Properties

P-69 Admin Executor address change scoping

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_GC1_only_set_admin_execute_can_change_admin_executor	Verified	*admin executor address can only be changed as a result of setAdminExecutor call*	*Report*

P-70 Governance address scoping

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_GC2_only_set_governance_can_set_governance	Verified	*governance address can only be changed as a result of setGovernance (or emergenceReset, see the "Emergency mode deactivation") call*	Report

P-71 After submit delay change scoping

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_GC3_only_set_after_submit_delay_can_set_delay	Verified	*post-submit delay can only be changed as a result of setAfterSubmitDelay call*	Report

P-72 After schedule delay change scoping

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_GC4_only_set_after_schedule_	Verified	*post-schedule delay can only be changed as a result of setAfterScheduleDelay call*	Report

delay_can_set_delay			

P-73 Admin Executor function call privilege

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_GC5_only_admin_executor_can_call_some_functions	Verified	*setGovernance, setAdminExecutor, setAfterSubmitDelay, setAfterScheduleDelay, transferExecutorOwnership cannot be called by any address other than the admin executor address*	*Report*

EPT General Mechanics

Spec General Assumptions

- Same as general verification assumptions

Spec Properties

P-74 Only governance can call submit

Status: Verified

Rule Name	Status	Description	Link to rule report

EPT_GM2_only_governance_can_call_submit	Verified	*submit cannot be called by any address other than the governance address*	Report

P-75 Only governance can call schedule

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_GM3_only_governance_can_call_schedule	Verified	*schedule cannot be called by any address other than the governance address*	Report

P-76 A non-scheduled proposal cannot be executed.

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_GM4_non_scheduled_proposal_cant_be_executed	Verified	*a non-scheduled proposal cannot be executed or emergency executed at any point in time*	Report

P-77 Re-execution is not possible

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_GM5_executed_proposal_cant_be_executed	Verified	*an executed proposal cannot be re-executed or emergency re-executed at any point in time*	*Report*

P-78 Proposal post-submit delay enforced

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_GM6_cant_schedule_before_post_submit_delay	Verified	*a proposal cannot be scheduled before the post-submit delay passes since its submission*	*Report*

P-79 Proposal post-schedule delay enforced

Status: Verified

Rule Name	Status	Description	Link to rule report
EPT_GM7_cant_execute_before_post_schedule_delay	Verified	*a scheduled proposal cannot be executed before the post-schedule delay passes since its scheduling*	*Report*

Timelocked Governance

Spec General Assumptions

- Same as general verification assumptions

Spec Properties

P-80 Only governance can call submitProposal

Status: Verified

Rule Name	Status	Description	Link to rule report
TG1_only_governance_can_submit_proposal	Verified	*submitProposal cannot be called by any address other than the TimelockedGovernance.GOVERNANCE() address.*	Report

P-81 Only governance can call cancelAllPendingProposals

Status: Verified

Rule Name	Status	Description	Link to rule report
TG2_only_governance_can_cancel_proposals	Verified	*cancelAllPendingProposals cannot be called by any address other than the TimelockedGovernance.GOVERNANCE() address.*	Report

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.