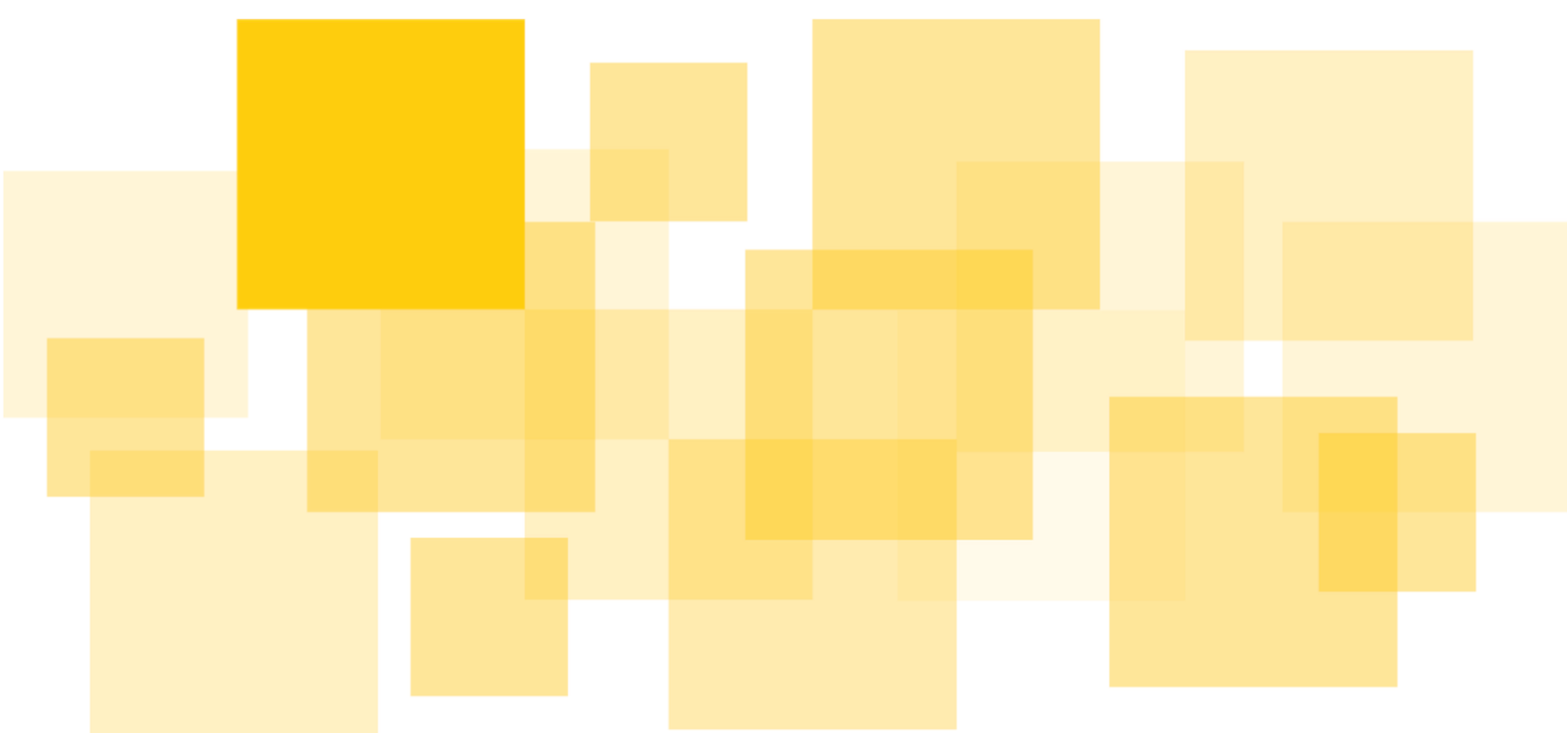


# Design Review

---

## Lido (Dual Governance)

Delivered: Apr 30, 2024



Prepared for Lido by





# Table of Contents

---

- [Disclaimer](#)
- [Summary and Scope](#)
- [Timing Analysis of Dual Governance States](#)
- [Overall Bounds on Proposal Execution](#)
- [Key Properties of Dual Governance](#)
- [Proofs](#)
  - [Veto Signalling Maximum Timelock](#)
  - [Staker Reaction Time](#)



## Disclaimer

---

Not addressed by client

This report does not constitute legal or investment advice. You understand and agree that this report relates to new and emerging technologies and that there are significant risks inherent in using such technologies that cannot be completely protected against. While this report has been prepared based on data and information that has been provided by you or is otherwise publicly available, there are likely additional unknown risks which otherwise exist. This report is also not comprehensive in scope, excluding a number of components critical to the correct operation of this system. This report is for informational purposes only and is provided on an "as-is" basis and you acknowledge and agree that you are making use of this report and the information contained herein at your own risk. The preparers of this report make no representations or warranties of any kind, either express or implied, regarding the information in or the use of this report and shall not be liable to you or any third parties for any acts or omissions undertaken by you or any third parties based on the information contained herein.



## Summary and Scope

---

Not addressed by client

This design review of the Lido Dual Governance protocol was conducted by Runtime Verification, Inc. from April 02, 2024, until April 30, 2024. The focus of the review was on analyzing the Dual Governance mechanism as described and specified in the following two documents:

- [Dual Governance mechanism design](#)
- [Dual Governance specification](#)

The initial version of the design documents was fixed at the following commit:

<https://github.com/lidofinance/dual-governance/tree/795c43e1faad2fcd4eb4320ee74dfd85119469b5/docs>. However, further changes were made to the documents as the client identified and fixed issues with the protocol. These changes were also reviewed as part of this design review, and this report takes into account the version of the protocol as of the following commit:

<https://github.com/lidofinance/dual-governance/tree/066810d3ca52b01f3eb6172c1a7ff9b498b0c9ad/docs>. The individual changes are described in the changelog at the end of the `mechanism.md` file: <https://github.com/lidofinance/dual-governance/blob/066810d3ca52b01f3eb6172c1a7ff9b498b0c9ad/docs/mechanism.md#changelog>.

This report is organized as follows. The **Timing Analysis of Dual Governance States** section provides upper and lower bounds on the duration of each state in the protocol. These bounds are used in the following section, **Overall Bounds on Proposal Execution**, to also give upper and lower bounds on the time between proposal submission and when this proposal becomes executable. These two sections provide certain guarantees against two diametrically opposed types of attacks: attacks that block execution of legitimate proposals, and attacks that prevent blocking malicious proposals. The **Key Properties of Dual Governance** outlines other important properties of the protocol, including those that must be guaranteed by the implementation and those that emerge from the mechanism design. Finally, the **Proofs** section provides formal proofs for some of the properties mentioned in the previous sections.

# Timing Analysis of Dual Governance States

Not addressed by client

There are two opposing ways in which a malicious adversary might attack the protocol by manipulating proposal execution delays:

1. Delay the execution of a legitimate proposal for a significant amount of time or, in the worst case, indefinitely.
2. Trigger an early execution of a malicious proposal, without giving the stakers the chance to delay it or exit the protocol.

As these two attack vectors are diametrically opposed, any change in the protocol made to prevent one has the potential of enabling the other. Therefore, it's important to know both the minimum and maximum time that the execution of a proposal can be delayed for in the current version of the protocol. The first step is to analyze how much time can be spent inside each Dual Governance state, since the current state is one of the main factors that determine whether a proposal can be executed.

Below, we give upper and lower bounds on the time spent in each state, given as the difference between the time the state was entered ( $t_{enter}$ ) and exited ( $t_{exit}$ ).

**Note:** For simplicity, this analysis assumes that transitions happen immediately as soon as they are enabled. Since in practice they need to be triggered by a call to `activateNextState`, it's possible for there to be a delay between when the transition becomes enabled and when it actually happens. However, we can assume this delay will be small since any interested agent can call the function as soon as it becomes possible.

## Normal

The Normal state can only transition to the Veto Signalling state, and this transition happens immediately as soon as the rage quit support surpasses `FirstSealRageQuitSupport` ( $R_1$ ). On the other hand, if this never happens the protocol can remain in the Normal state indefinitely. Therefore, the time between activating the Normal state and transitioning to the Veto Signalling state can have any duration:

$$0 \leq t_{exit} - t_{enter} < \infty$$

## Veto Signalling

Once the Veto Signalling state is entered, it can be exited in two ways: either to the Rage Quit state or the Veto Cooldown state. It also can enter and exit the Deactivation sub-state, making this the hardest state to analyze.

### To Veto Cooldown

While in the Veto Signalling state, the protocol can enter and exit the Deactivation sub-state depending on the current value of the dynamic timelock duration  $T_{lock}(R)$ , a monotonic function on the current rage quit support  $R$ .

When first entering the Veto Signalling state, and again whenever the Deactivation sub-state is exited, there is a waiting period of `VetoSignallingMinActiveDuration` ( $T_{min}^{Sa}$ ) when the Deactivation sub-state cannot be entered. Outside of this window (as long as a rage quit is not triggered), the protocol will be in the Deactivation sub-state if the time  $\Delta t$  since entering the Veto Signalling state is greater than the current value of  $T_{lock}(R)$ , and will be in the Veto Signalling parent state otherwise. If the Deactivation sub-state is not exited within `VetoSignallingDeactivationMaxDuration` ( $T_{max}^{SD}$ ) of being entered, it transitions to the Veto Cooldown state.

With this, we can calculate bounds on the time spent in the Veto Signalling state (including the Deactivation sub-state) before transitioning into Veto Cooldown:

- For the lower bound, the earliest we can transition to the Deactivation sub-state is  $T_{min}^{Sa} + 1$  after Veto Signalling is entered, and then the transition to Veto Cooldown happens  $T_{max}^{SD} + 1$  after that, giving us  $T_{min}^{Sa} + T_{max}^{SD} + 2$ .
- For the upper bound, we can use the insight that, if  $R_{max}$  is the highest rage quit support during the time we are in the Veto Signalling state, then it's impossible to exit the Deactivation sub-state (without triggering a rage quit) after  $T_{lock}(R_{max})$  has passed since entering Veto Signalling (see "Veto Signalling Maximum Timelock" in the "Proofs" section for details). Therefore, for a given  $R_{max}$ , the longest delay happens in the following scenario:
  1.  $R_{max}$  is locked in escrow, making the dynamic timelock duration  $T_{lock}(R_{max})$ .
  2. Shortly before  $\Delta t = T_{lock}(R_{max})$  has passed, the rage quit support decreases, and the Deactivation sub-state is entered.

3. At exactly  $\Delta t = T_{lock}(R_{max})$ , the rage quit support returns to  $R_{max}$ , and the Deactivation sub-state is exited.
4. At  $\Delta t = T_{lock}(R_{max}) + T_{min}^{Sa} + 1$ , the waiting period ends and the Deactivation sub-state is entered again.
5. At  $\Delta t = T_{lock}(R_{max}) + T_{min}^{Sa} + 1 + T_{max}^{SD} + 1$ , the state transitions to Veto Cooldown.

In summary, the above analysis gives us the following bounds:

$$T_{min}^{Sa} + T_{max}^{SD} + 2 \leq t_{exit} - t_{enter} \leq T_{lock}(R_{max}) + T_{min}^{Sa} + T_{max}^{SD} + 2$$

Note that the maximum value of  $T_{lock}(R)$  is `DynamicTimelockMaxDuration` ( $L_{max}$ ), so the upper bound can be at most  $L_{max} + T_{min}^{Sa} + T_{max}^{SD} + 2$ . However, writing it in terms of  $R_{max}$  highlights an important security property: the delay in deactivating the Veto Signalling state depends only on the *highest* value of the rage quit support, and cannot be increased further by locking and unlocking funds in the signalling escrow at different times. In other words, the amount of delay an attacker is able to achieve is limited by the amount of stETH they control.

## To Rage Quit

The Veto Signalling state can transition to the Rage Quit state at any point after  $L_{max}$  has passed, as long as the rage quit support surpasses `SecondSealRageQuitSupport` ( $R_2$ ). This gives us a lower bound of  $L_{max}$ . For the upper bound, we can adapt the analysis of the Veto Cooldown transition above. Note that if  $R_{max} = R_2$ , it's possible to delay the transition to the Veto Cooldown state for the maximum time of  $L_{max} + T_{min}^{Sa} + T_{max}^{SD} + 2$  before triggering a rage quit at the last possible moment by increasing the rage quit support above  $R_2$ . Therefore,

$$L_{max} < t_{exit} - t_{enter} \leq L_{max} + T_{min}^{Sa} + T_{max}^{SD} + 2$$

## Veto Cooldown

Depending on whether the rage quit support is above or below  $R_1$ , the Veto Cooldown state can transition to the Veto Signalling state or the Normal state, but regardless this transition always happens after `VetoCooldownDuration` ( $T^C$ ) has passed. Therefore, the time between entering and exiting the state is always the same:

$$t_{exit} - t_{enter} = T^C + 1$$

## Rage Quit

The Rage Quit state differs from the others due to having a dependence on a mechanism external to the Dual Governance protocol, namely the withdrawal procedure for unstaking ETH. After transitioning to the Rage Quit state, anyone can call the `requestNextWithdrawalsBatch` function, sending a portion of the funds in the rage quit escrow to the Lido Withdrawal Queue. Once a withdrawal request is finalized, anyone can transfer the ETH from the queue to the rage quit escrow by calling the `claimNextWithdrawalsBatch` function. After all the withdrawals are claimed, a period lasting `RageQuitExtensionDelay` ( $T^R$ ) starts, where any stakers who had submitted Withdrawal NFTs as rage quit support and haven't done so already can claim those as well before the Rage Quit state is exited.

Therefore, if  $T^W$  is the time from when the Rage Quit state is entered until the last withdrawal request is claimed, then the total duration of the Rage Quit state is

$$t_{exit} - t_{enter} = T^W + T^R + 1$$

Unlike other values in this analysis, the time duration  $T^W$  cannot be deterministically predicted or bounded based on the internal state of the Dual Governance protocol, as withdrawal time is dependent on a number of factors related to Ethereum's validator network and other parts of the Lido system. However, triggering a rage quit in bad faith would come at a higher cost than normal to an attacker, as it would not only require them to remove their stake from the protocol, but also keep their ETH locked in the rage quit escrow until a dynamic `RageQuitEthClaimTimelock` has passed after exiting the Rage Quit state.



# Overall Bounds on Proposal Execution

Not addressed by client

Using the bounds from the previous section on the duration of each Dual Governance state, we can now set bounds on the total time between when a proposal is submitted ( $t_{sub}$ ) and when it becomes executable ( $t_{exe}$ ). For this analysis, we need to take into account the following rules of the protocol:

1. Proposal submission is only allowed in the Normal, Veto Signalling (only the parent state, not the Deactivation sub-state) and Rage Quit states.
2. Proposal execution is only allowed in the Normal and Veto Cooldown states.
3. Regardless of state, a proposal can only be executed after `ProposalExecutionMinTimelock` ( $T_{min}$ ) has passed since its submission.
4. In the Veto Cooldown state, a proposal can only be executed if it was submitted before the last time the Veto Signalling state was entered.

Rule 4 is meant to guarantee that if a proposal is submitted during the Veto Signalling or Rage Quit states, the stakers will have the time to react and the benefit of a full Veto Signalling dynamic timelock before the proposal becomes executable.

## Accounting for Rage Quits

Note that it is technically possible to delay execution of a proposal indefinitely if the protocol transitions continuously between the Veto Signalling and Rage Quit states. However, as mentioned above, triggering a Rage Quit is unlikely to be cost-efficient to an attacker. Furthermore, doing this repeatedly is even more unlikely to be viable, as after exiting the Rage Quit state the funds remain locked in the rage quit escrow for `RageQuitEthClaimTimelock`, which starts as a lengthy duration (60 days with the current proposed values) and increases quadratically with each subsequent rage quit until the protocol returns to the Normal state.

With this in mind, in the rest of this section we consider the bounds on proposal execution assuming no rage quit is triggered. If the Rage Quit state is entered, every entry will delay proposal execution for an additional  $T^W + T^R + 1$  (keeping in mind that  $T^W$  is a non-deterministic duration that depends on external factors and might be different each time the rage quit is triggered), plus a further delay between  $T_{min}^{Sa} + T_{max}^{SD} + 2$  and  $T_{lock}(R_{max}) + T_{min}^{Sa} + T_{max}^{SD} + 2$  if the Rage Quit state then transitions back to Veto Signalling. However, note

that, after this re-entry to Veto Signalling, if the protocol then transitions to Veto Cooldown, any proposals submitted during the previous Rage Quit state or the Veto Signalling state before that will become executable immediately, without needing another round of Veto Signalling.

More precisely, the following bounds apply to any proposal submitted in the Rage Quit state or the preceding Veto Signalling state, for the time between when the Rage Quit state is exited ( $t_{exit}^R$ ) and when the proposal becomes executable (assuming no further rage quit happens and the minimum timelock  $T_{min}$  has already passed):

- If the Rage Quit state exits to Veto Cooldown and then Normal (becomes executable as soon as the Normal state is entered):

$$t_{exe} - t_{exit}^R = T^C + 1$$

- If the Rage Quit state exits to Veto Cooldown, then Veto Signalling and Veto Cooldown again (becomes executable as soon as the Veto Cooldown state is entered for the second time):

$$T^C + T_{min}^{Sa} + T_{max}^{SD} + 3 \leq t_{exe} - t_{exit}^R \leq T^C + T_{lock}(R_{max}) + T_{min}^{Sa} + T_{max}^{SD} + 3$$

- If the Rage Quit state exits to Veto Signalling and then Veto Cooldown (becomes executable as soon as the Veto Cooldown state is entered):

$$T_{min}^{Sa} + T_{max}^{SD} + 2 \leq t_{exe} - t_{exit}^R \leq T_{lock}(R_{max}) + T_{min}^{Sa} + T_{max}^{SD} + 2$$

## Proposals Submitted in the Normal State

If a proposal is submitted in the Normal state and no transition happens before then, it become executable as soon as  $T_{min} + 1$  passes:

$$t_{exe} - t_{sub} = T_{min} + 1$$

On the other hand, if the protocol transitions to Veto Signalling before this time due to rage quit support surpassing  $R_1$ , it will become subject to the Veto Signalling dynamic timelock. The shortest possible time for execution in this case happens if the transition happens immediately after the proposal is submitted (at the same timestamp), and soon after that the rage quit support drops below  $R_1$  again, making the dynamic timelock 0. In this scenario, the Deactivation sub-state will be entered at  $T_{min}^{Sa} + 1$ , and then exited to the Veto Cooldown state

at  $T_{max}^{SD} + 1$ , giving the previously-stated  $T_{min}^{Sa} + T_{max}^{SD} + 2$  lower bound on the duration of the Veto Signalling state. Note, however, that it's possible, depending on the parameter values, that at this point the minimum timelock  $T_{min}$  hasn't passed, so the true lower bound is the highest between the minimum timelock and the minimum duration of the Veto Signalling state:

$$\max\{T_{min} + 1, T_{min}^{Sa} + T_{max}^{SD} + 2\} \leq t_{exe} - t_{sub}$$

Note that for the current proposed values, the minimum Veto Signalling duration is slightly higher:

- $T_{min}$  is 3 days.
- $T_{min}^{Sa}$  is 5 hours.
- $T_{max}^{SD}$  is 3 days.

For the upper bound, the longest possible delay happens when the Veto Signalling state is entered at  $T_{min}$  (the last possible moment before the proposal becomes executable in the Normal state) and lasts as long as possible ( $T_{lock}(R_{max}) + T_{min}^{Sa} + T_{max}^{SD} + 2$ , according to our previous analysis):

$$t_{exe} - t_{sub} \leq T_{min} + T_{lock}(R_{max}) + T_{min}^{Sa} + T_{max}^{SD} + 2$$

**Note:** With the current proposed values for the parameters, we have the guarantee that  $T_{min}$  will have expired by the time the Veto Cooldown state is entered. However, if  $T_{min}$  were greater than  $T_{min}^{Sa} + T_{max}^{SD}$ , it would be possible to enter and exit the Veto Signalling state before the minimum timelock expired. If it were greater than  $T_{min}^{Sa} + T_{max}^{SD} + T^C$ , it would even be possible to exit Veto Cooldown and re-enter Veto Signalling before it expired. Nevertheless, the above upper bound would still work in those cases. In the first case, the proposal would become executable in the Veto Cooldown state at  $T_{min} + 1$ , which is less than the upper bound. In the second case, the Veto Signalling state would have to be re-entered at  $T_{min}$  at the latest, giving rise to the same bound as above.

## Proposals Submitted in the Veto Signalling State

Proposals submitted while in the Veto Signalling state are not immediately executable after transitioning to Veto Cooldown. Instead, they will either become executable when Veto Cooldown transitions to the Normal state, or, if it transitions back to Veto Signalling instead, once it exits that state again and returns to Veto Cooldown.

In the first case, the shortest time between proposal submission and execution happens in the following scenario:

1. The proposal is submitted immediately before entering the Deactivation sub-state (recall that submissions are only allowed in the parent state).
2. After  $T_{max}^{SD} + 1$ , the Deactivation sub-state transitions to Veto Cooldown.
3. After  $T^C + 1$ , Veto Cooldown transitions to the Normal state and the proposal becomes executable.

Meanwhile, the longest time to execution happens in the following scenario:

1. The proposal is submitted as soon as the Veto Signalling state is first entered.
2. The longest possible duration of  $T_{lock}(R_{max}) + T_{min}^{Sa} + T_{max}^{SD} + 2$  passes before transitioning to Veto Cooldown.
3. After  $T^C + 1$ , Veto Cooldown transitions to the Normal state and the proposal becomes executable.

However, if either of these scenarios takes less time than  $T_{min}$  (again, not the case for the current proposed values), then we'll have to wait for the minimum timelock to pass before executing the proposal, giving us the following final bounds:

$$\max\{T_{min} + 1, T_{max}^{SD} + T^C + 2\} \leq t_{exe} - t_{sub}$$

$$t_{exe} - t_{sub} \leq \max\{T_{min} + 1, T_{lock}(R_{max}) + T_{min}^{Sa} + T_{max}^{SD} + T^C + 3\}$$

In the second case, where Veto Cooldown transitions back to Veto Signalling instead of the Normal state, we need to add the duration of the Veto Signalling state again to the bounds. For the lower bound, we add the minimum duration of  $T_{min}^{Sa} + T_{max}^{SD} + 2$ , giving us  $T_{min}^{Sa} + 2T_{max}^{SD} + T^C + 4$ . For the upper bound, we add the maximum duration of  $T_{lock}(R'_{max}) + T_{min}^{Sa} + T_{max}^{SD} + 2$ , where  $R'_{max}$ , the maximum rage quit support during the second time through the Veto Signalling state, might be different from  $R_{max}$ . This leaves us with the following bounds:

$$\max\{T_{min} + 1, T_{min}^{Sa} + 2T_{max}^{SD} + T^C + 4\} \leq t_{exe} - t_{sub}$$

$$t_{exe} - t_{sub} \leq \max\{T_{min} + 1, T_{lock}(R_{max}) + T_{lock}(R'_{max}) + 2T_{min}^{Sa} + 2T_{max}^{SD} + T^C + 5\}$$



# Key Properties of Dual Governance

---

Not addressed by client

## Implementation Properties

These are properties that must be guaranteed by the implementation of the contracts that comprise the Dual Governance mechanism.

### DualGovernance

- Proposals cannot be executed in the Veto Signalling (both parent state and Deactivation sub-state) and Rage Quit states.
- Proposals cannot be submitted in the Veto Signalling Deactivation sub-state or in the Veto Cooldown state.
- If a proposal was submitted after the last time the Veto Signalling state was activated, then it cannot be executed in the Veto Cooldown state.
- One rage quit cannot start until the previous rage quit has finalized. In other words, there can only be at most one active rage quit escrow at a time.

### EmergencyProtectedTimelock

- A proposal cannot be scheduled for execution before at least `ProposalExecutionMinTimelock` has passed since its submission.
- A proposal cannot be executed until the emergency protection timelock has passed since it was scheduled.
- The emergency protection timelock is greater than 0 if and only if the protocol is in protected deployment mode.

### Escrow

- Ignoring imprecisions due to fixed-point arithmetic, the rage quit support of an escrow is equal to  $\frac{(S+W+U+F)}{(T+F)}$  where
  - $S$  is the ETH amount locked in the escrow in the form of stETH.
  - $W$  is the ETH amount locked in the escrow in the form of wstETH.
  - $U$  is the ETH amount locked in the escrow in the form of unfinalized Withdrawal NFTs.
  - $F$  is the ETH amount locked in the escrow in the form of finalized Withdrawal NFTs.

- $T$  is the total supply of stETH.
- The amount of each token accounted for in the above calculation must be less than or equal to the balance of the escrow in the token.
- It's not possible to lock funds in or unlock funds from an escrow that is already in the rage quit state.
- An agent cannot unlock their funds from the signalling escrow until `SignallingEscrowMinLockTime` has passed since this user last locked funds.

## Protocol Properties

These are emergent properties that are derived from the design of the protocol, and give guarantees of protection against certain attacks.

- 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.

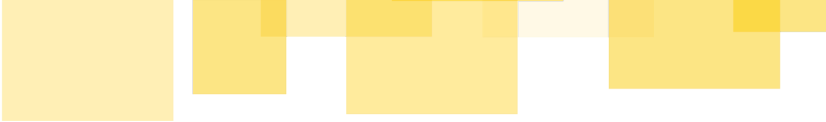
The proof for this property is presented in the "Proofs" section, under "Staker Reaction Time". However, note that it depends on the assumption that the minimum duration of the Veto Signalling state is greater than or equal to `ProposalExecutionMinTimelock` (which, as pointed out in the previous section, is true for the current proposed parameter values).

- It's not possible to prevent a proposal from being executed indefinitely without triggering a rage quit.

This property is guaranteed by the upper bounds on proposal execution presented in the previous section, "Overall Bounds on Proposal Execution".

- It's not possible to block proposal submission indefinitely.

This property is guaranteed by the fact that the only states that forbid proposal submission (Veto Cooldown and Veto Signalling Deactivation) have a fixed maximum duration, and that they cannot transition back-and-forth without a period in between when proposal submission is allowed: if the Veto Cooldown state transitions to the Veto Signalling state, it must remain in the parent state for at least `VetoSignallingMinActiveDuration` before it can transition to the Deactivation sub-state.

- 
- Until the Veto Signalling Deactivation sub-state transitions to Veto Cooldown, there is always a possibility (given enough rage quit support) of cancelling Deactivation and returning to the parent state (possibly triggering a rage quit immediately afterwards).

This property is guaranteed by the transition function of the Deactivation sub-state, which always exits to the parent state when the current rage quit support is greater than `SecondSealRageQuitSupport`, regardless of how much time has passed. However, if `DynamicTimelockMaxDuration` has passed since the Veto Signalling state was entered, it will immediately trigger a transition from Veto Signalling to the Rage Quit state.

# Proofs

Not addressed by client

## Veto Signalling Maximum Timelock

### Property

Suppose that at time  $t_{act}^S$  the Dual Governance protocol enters the Veto Signalling state. Then, if a rage quit isn't triggered, and assuming that state transitions are activated as soon as they becomes enabled, the protocol will transition to the Veto Cooldown state at a time  $t_{act}^C > t_{act}^S$  such that

$$t_{act}^C \leq t_{act}^S + T_{lock}(R_{max}) + T_{min}^{Sa} + T_{max}^{SD} + 2$$

where  $R_{max}$  is the maximum rage quit support between  $t_{act}^S$  and  $t_{act}^C$ .

### Proof

**Observation 1:** First note that if we are in the Deactivation sub-state at any time after  $t_{act}^S + T_{lock}(R_{max})$ , we will not exit the sub-state until we transition to Veto Cooldown. To exit back to the parent state, one of the following would have to be true for some  $t > t_{act}^S + T_{lock}(R_{max})$ :

1.  $t \leq t_{act}^S + T_{lock}(R(t))$ . Since  $R_{max} \geq R(t)$  for all  $t$  between  $t_{act}^S$  and  $t_{act}^C$ , and  $T_{lock}(R)$  is monotonic, this is not possible once  $t > t_{act}^S + T_{lock}(R_{max})$ .
2.  $R(t) > R_2$ . This would imply  $R_{max} \geq R(t) > R_2$ , and therefore  $T_{lock}(R_{max}) = L_{max}$ . Since  $t - t_{act}^S > T_{lock}(R_{max}) = L_{max}$  and  $R(t) > R_2$ , after exiting the Deactivation sub-state a rage quit would be immediately triggered. Since we are only considering scenarios where no rage quit happens, this case is also impossible.

Now, let  $t_{act}^{SD}$  be the last time the Deactivation sub-state is entered before  $t_{act}^C$ . We will first prove that  $t_{act}^{SD} \leq t_{act}^S + T_{lock}(R_{max}) + T_{min}^{Sa} + 1$ :

- **Case 1:** If  $t_{act}^{SD} \leq t_{act}^S + T_{lock}(R_{max}) + 1$ , we are done.
- **Case 2:** Otherwise,  $t_{act}^S + T_{lock}(R_{max}) + 1 < t_{act}^{SD}$ .
  - Then, from Observation 1 above, at  $t_1 = t_{act}^S + T_{lock}(R_{max}) + 1$  we cannot be in the Deactivation sub-state (since we wouldn't be able to exit the sub-state to enter again at  $t_{act}^{SD}$ ).



- Since this is the case even though  $t_1 - t_{act}^S > T_{lock}(R(t_1))$ , it must be because  $t_1 - \max\{t_{act}^S, t_{react}^S\} \leq T_{min}^{Sa}$ , where  $t_{react}^S < t_1$  was the last time the Deactivation sub-state was exited, or 0 if it has never been entered (note that  $t_1$  must be strictly greater than  $t_{react}^S$ , since it would be impossible to transition back to the parent state at  $t_{act}^S + T_{lock}(R_{max}) + 1$ ).
- In this case, as  $t - t_{act}^S > T_{lock}(R(t))$  will remain true for any future  $t > t_1$ , the transition at  $t_{act}^{SD}$  must happen as soon as  $t - \max\{t_{act}^S, t_{react}^S\} > T_{min}^{Sa}$  becomes true.
- Since  $t_1$  is strictly greater than  $\max\{t_{act}^S, t_{react}^S\}$ , the latest this can happen is at  $t_1 + T_{min}^{Sa} = t_{act}^S + T_{lock}(R_{max}) + T_{min}^{Sa} + 1$ .

Finally, since the Deactivation sub-state does not return to the parent state after  $t_{act}^{SD}$ , and no rage quit is triggered, it will transition to Veto Cooldown as soon as  $t - t_{act}^{SD} > T_{max}^{SD}$ .

Therefore,  $t_{act}^C = t_{act}^{SD} + T_{max}^{SD} + 1 \leq t_{act}^S + T_{lock}(R_{max}) + T_{min}^{Sa} + T_{max}^{SD} + 2$ .

### Caveats

- This proof assumes that the transitions at  $t_{act}^{SD}$  and  $t_{act}^C$  happen immediately as soon as they are enabled. Any delay  $d$  in performing either of these transitions gets added to the upper bound. For instance, if the last transition to the Deactivation sub-state is only performed at  $t_{act}^{SD} + d$ , then the earliest that Veto Cooldown can be entered is  $t_{act}^{SD} + d + T_{max}^{SD} + 1$ . On the other hand, any delay in performing previous transitions between the Deactivation sub-state and the parent state (before the last one at  $t_{act}^{SD}$ ) does not increase the upper bound, since it does not change the time  $t_0 + T_{lock}(R_{max}) + 1$ .

## Staker Reaction Time

The following property can be interpreted to say that, regardless of the state in which a proposal is submitted, if the stakers are able to amass and maintain rage quit support  $R_{min}$  before the `ProposalExecutionMinTimelock` expires, they can extend the timelock to at least  $T_{lock}(R_{min})$ .

### Property

Suppose that a proposal is submitted at time  $t_{prop}$ , and let  $t_1$  and  $t_2$  be such that

1.  $t_{prop} \leq t_1 \leq t_{prop} + T_{min} \leq t_2$ , where  $T_{min}$  is `ProposalExecutionMinTimelock`.

2. Between  $t_1$  and  $t_2$ , the rage quit support is never lower than some value  $R_{min}$ .
3.  $t_2 \leq t_{prop} + T_{lock}(R_{min})$ .

Then, assuming that  $T_{min}$  is less than the minimum duration of the Veto Signalling state, the proposal cannot be executed at any time less than or equal to  $t_2$ .

### Proof

First, note that if  $R_{min} \leq R_1$ , then  $T_{lock}(R_{min}) = 0$  and  $t_2 = t_{prop}$ . Since this cannot be the case when  $t_{prop} + T_{min} \leq t_2$ , we only need to consider cases where  $R_{min} > R_1$ .

At  $t_{prop}$ , the protocol must be in one of the three states that allow proposal submission. We'll consider each case individually.

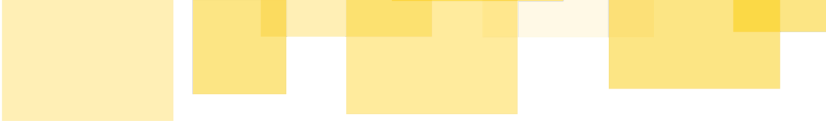
**Normal state:** Since  $T_{min}$  is less than the minimum duration of the Veto Signalling state, immediately before  $t_1$  the protocol must be in either the Normal state or the Veto Signalling state. Regardless, since  $R(t_1) \geq R_{min} > R_1$ , at  $t_1$  the protocol will be in the Veto Signalling state. Then, given that, for every  $t_1 \leq t \leq t_2$ ,

- $t \leq t_{prop} + T_{lock}(R_{min})$
- $t_{prop} \leq t_{act}^S$
- $T_{lock}(R_{min}) \leq T_{lock}(R(t))$

we have that  $t \leq t_{act}^S + T_{lock}(R(t))$ , or  $t - t_{act}^S \leq T_{lock}(R(t))$ . Therefore, the protocol cannot be in the Deactivation sub-state at any  $t$  between  $t_1$  and  $t_2$ . Since  $T_{lock}(R(t)) \leq L_{max}$ , the Rage Quit state cannot be entered either. Therefore, the Veto Signalling state cannot be exited, and consequently the proposal is not executable during this time.

**Veto Signalling state:** If immediately before  $t_1$  the protocol is either in the Normal state or the Veto Signalling state, the same reasoning as the previous case applies. Otherwise, it must be in either the Veto Cooldown or Rage Quit states. Since  $T_{min}$  is less than the minimum duration of the Veto Signalling state, it cannot be the case that the Veto Signalling state was entered and exited again between  $t_{prop}$  and  $t_1$ . Therefore, at  $t_1$  the proposal is not executable in either the Veto Cooldown or Rage Quit state. Then, there are two cases:

- If we remain in the same state until  $t_2$ , the proposal remains not executable.
- If we transition to another state before  $t_2$ , it will be to the Veto Signalling state, since  $R_{min} > R_1$ . Using the same reasoning as above, we can conclude that we cannot exit the



Veto Signalling state for any  $t_1 \leq t \leq t_2$ , and therefore the proposal will remain not executable until at least  $t_2$  as well.

**Rage Quit state:** The same reasoning as the previous case applies.