certora

Security Assessment & Formal Verification Report

Squads Smart Account Program v0.1

Jan 2025

Prepared for Squads

Table of contents

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Squads Smart Account Program v0.1	Squads-Protocol/smart-account-program	936c88c3e8649107d2f978db84db4f89e913730f	Solana

Project Overview

This document describes the specification and verification of **Squads** using the Certora Prover and manual code review findings. The work was undertaken from **Jan 7** to **Jan 28, 2025**

The following contract list is included in our scope:

```
squads_smart_account_program/*
```

The Certora Prover demonstrated that the implementation of the **Solana** 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 Solana contracts**.** The Certora team discovered bugs in the Solana contracts code during the verification process and the manual audit, as listed on the following page.

Protocol Overview

The smart account program is a fork of the now immutable Squads Protocol v4 program. It builds on all the Squads Protocol v4 functionality and adds synchronous methods to aid in the UX of multi-signer and gas-abstracted transactions. Other minor changes include the ability to add an expiration period to spending limits.

The protocol acts as a programmable smart account layer that enables complex multi-party account management and transaction execution flows, aka a "Multisig." Members of the multisig "Settings" can vote to approve or reject any set of arbitrary transactions or transactions altering the Multisig configuration.

Findings Summary

The table below summarizes the review's findings, including details on type and severity.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	1	1	0
Low	1	1	0
Informational	5	5	4
Total	7	7	4

Severity Matrix

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

Detailed Findings

ID	Title	Severity	Status
M-01	Batch proposal can be prematurely activated before all transactions are added	Medium	Acknowledged
L-01	Smart account allows adding invalid signer accounts	Low	Acknowledged
I-01	Redundant checks	Informational	Fixed
I-02	realloc_if_needed function refactor	Informational	Fixed
I-03	Terminology and Typos	Informational	Fixed
I-04	Mismatched numerical types between threshold and num_signers parameters	Informational	Acknowledged
I-05	Refactor checks on proposal status in transaction_close.rs	Informational	Fixed

Medium Severity Issues

M-01 Batch proposal can be prematurely activated before all transactions are added		
Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: batch_add_transaction.rs activate_proposal.rs	Status: Acknowledged	

Description: A batch creator can have their batch prematurely activated by anyone with the Initiate permission, preventing the batch creator from adding all the intended transactions to said batch. This could be very detrimental in time-sensitive scenarios where misaligned incentives exist between members of two settings. This would enable the adversarial setting member to cause irreparable damage before they could be removed.

For example, Alice, the batch creator, needs to close a leveraged position on behalf of settings. However, Bob has a feud with Alice and decides to block her from closing the position, which gets liquidated.

Recommendations: We recommend allowing the batch creator to have the option to pre-determine the batch size and require this pre-determined value to be equal to the actual batch size before the proposal can be activated.

Customer's response: Acknowledged. In any case, where this issue arises, the malicious member will simply be kicked out.

Low Severity Issues

L-01 Smart account allows adding invalid signer accounts		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: settings.rs authority_settings_transaction_execute.rs	Status: Acknowledged	

Description: The smart account's add_signer() function does not validate whether a pubkey submitted as a new signer corresponds to a valid account. This could lead to a scenario where an invalid signer is added followed by the removal of a valid signer, potentially resulting in the account having fewer than the required number of valid signers. Currently, the contract only checks for duplicate signers and performs reallocation without verifying account validity.

Recommendations: Consider implementing one of these approaches:

1. A signature from the new signer is required when adding them to verify the account exists and can be signed.
2. Implement a two-step process where a new signer is first proposed, and then that signer must call an acceptance function to complete the addition. This ensures the account exists and is controlled by someone who can sign transactions.

Note: The same consideration should apply to the archival_authority field when it is implemented—either requiring a signature or implementing a two-step acceptance process for the new authority.

Customer's response: Acknowledged. This is valid feedback, but in practice, requiring signatures to add members is too much overhead, especially when considering that the members may not be in the same place at the same time.

Informational Severity Issues

I-01. Redundant checks

Description:
Two instances of redundant checks have been identified in the transaction handling code:

1. In the key index validation logic, a saturating_sub() operation is used where a normal subtraction would suffice, since the condition key_index >= num_signers already ensures no underflow can occur.
2. In the buffer size validation, checking buffer.len() against MAX_BUFFER_SIZE is redundant since we already verify self.final_buffer_size as usize <= MAX_BUFFER_SIZE and self.buffer.len() <= self.final_buffer_size:

Customer's response: Item 1 - acknowledged. Item 2 - fixed in commit a0106ed.

Fix Review: Item 2 - fix confirmed.

I-02. realloc_if_needed function refactor

Description: The realloc_if_needed() function is duplicated between multisig and proposal account handling, with nearly identical logic for checking and performing account reallocation. The only difference is in how the required account size is calculated - Settings::size() vs Proposal::size(). This duplicated code could be refactored into a shared utility function to improve maintainability and reduce the chance of inconsistencies.

Consider extracting the common reallocation logic into a shared utility function that takes a generic size calculation function as a parameter. This would allow the same core reallocation logic to be reused while allowing different account types to specify their own size requirements. The function could be placed in a utils module.

Customer's response: Fixed in commit 40b170b.

Fix Review: Fix confirmed.

I-03. Terminology and Typos

Description:

1. Update Comments in **transaction.rs**

Replace all instances of "Vault" with "Smart Account" in the implementation comments. Change "MultisigTransaction" to "SettingsTransaction" in transaction handling comments. Update all references to "vault" in state validation comments to "smart account".

2. Update Comments in **settings_transaction.rs**

Replace all instances of "multisig" with "settings" in operation handling and state validation comments.

3. Update Comments in **settings.rs**

Replace four instances of "multisig" with "settings" in configuration, initialization, settings update, and transaction processing comments.

4. Update Comments in **proposal.rs**

In proposal processing, state checks, and requirements validation comments, replace all instances of "multisig" with "settings."

5. Update Comments in **batch_execute_transaction.rs**

Replace "Multisig" with "Settings" in batch execution comments.

6. Update Comments in **batch_add_transaction.rs**

Replace "multisig" with "settings" in validation comments.

7. Update Comments in **transaction_execute.rs**

Replace references to "vault" with "smart account" in execution logic and state validation comments.

8. Update Comments in **spending_limit.rs**

Replace references to "vault" with "smart account" in spending controls and limit check comments.

9. Update Struct Field Names in **transaction.rs**

Replace struct fields `vault_index` with `account_index` and `vault_bump` with `account_bump` in the Transaction struct.

10. Update Struct Field Names in **batch_execute_transaction.rs**

Replace the struct field `vault_index` with `account_index` in the BatchExecuteTransaction struct.

11. Update Struct Field Names in **batch.rs**

Replace the struct field `vault_index` with `account_index` in the Batch struct.

12. Update Enum Field Names in **settings_transaction.rs**

Replace field `vault_index` with `account_index` in the **SettingsAction** enum's **AddMember** variant.

13. Update Struct Field Names in **spending_limit.rs**

Replace the field `vault_index` with `account_index` in the SpendingLimit struct. Update struct documentation to replace "vault" with "smart_account".

14. Update Struct Field Names in **authority_spending_limit_add.rs**

Replace field `vault_index` with `account_index` in the **AddSpendingLimitArgs** struct.

These changes align the codebase with the updated terminology:

- "Multisig" → "Settings"
- "Vault" → "Smart Account"

- "Member" → "Signer"
- "configTransaction" → "settingsTransaction"
- "vaultTransaction" → "Transaction"
- "vaultIndex" → "accountIndex"

15. Update Comments in **settings.rs**

Replace `8 = Initiate` with `7 = Initiate` in settings invariant comments.

16. Update Comments in **synchronous_transaction_message.rs**.

Replace `accou` with `account` in the **SynchronousTransactionMessage** struct implementation on line 48.

Customer's response: Fixed in commit dafabb5.

Fix Review: Fix confirmed.

I-04. Mismatched numerical types between threshold and num_signers parameters

Description:

The threshold parameter in `MultisigCreate` is defined as a `u16,` while the `num_signers` field in `SyncTransactionArgs` uses a `u8`. While this mismatch does not pose a security risk due to practical runtime constraints limiting the number of signers to much lower, it represents an inconsistency in the type system that should be addressed to avoid potential problems during future development.

Customer's response: Acknowledged. We want to keep the synchronous transaction methods as lean as possible, so we will leave this as a u8 to save on 2 extra bytes.

I-05. Refactor checks on proposal status in transaction_close.rs

Description:
The functions in **transaction_close.rs** perform deserialization on the proposal, followed by checks on the status of the proposal. These checks on the proposal status are duplicated in functions `close_transaction`, `close_batch_transaction`, and `close_batch`.

Consider extracting these checks into a separate function that takes `Option<Proposal>` as an input. This refactoring would also benefit the formal verification.

Customer's response: Fixed in commit 936c88c.

Fix Review: Fix confirmed.

Formal Verification

Summary

The formal verification for Squads Smart Account Program v0.1 reestablished correctness of properties from the previous Certora audit, as well as verified new properties for the synchronous mode of executing transactions. The rules from the previous audit were renamed according to the new terminology.
The new properties for synchronous transactions are:

- P-01: Integrity of `sync_transaction` and `sync_settings_transaction`
- P-02: No double approve for `sync_transaction` and `sync_settings_transaction`
- P-03: Equivalence between synchronous and asynchronous mode of executing transactions. That is, 1) a successful sync transaction implies a proposal will be approved via asynchronous transaction, and 2) an approved proposal via asynchronous transaction implies a successful synchronous transaction.
- P-04: `sync_settings_transaction` preserves `Settings::invariant` and invalidates prior transactions on execution

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

1. Prover Configuration
    - The Solana contracts were compiled to SBFv1 using the Rust compiler version 1.75
    - The Solana version was solana-cli 1.18.16.
    - All loops were unrolled at most 3 iterations.

2. Main assumptions for verification
    - All verification harnesses call Squads instructions that take Anchor contexts as input. Thus, no serialization/deserialization code has been taken into account by the prover. The prover assumes that all Anchor accounts are initially filled with arbitrary values.
    - `Clock::get()` returns an arbitrary strictly monotonically increasing value
    - We do not consider CPIs as well as PDA computations for formal verification.
    - We do not consider code that performs account reallocation or rent computations
    - The `Vec` class used by `members`, `approved`, `rejected`, and `canceled` is replaced by our implementation `NoResizableVec` that assumes static vector size. To safely do so, the prover always establishes that the length of each of those vectors is less than their respective capacities, that is the vectors never need to grow. The correctness of `NoResizableVec` has been established separately.
    - Synchronous transactions are supplied with exactly two signers.

3. Code Modifications and refactoring
    - `close_transaction`, `close_settings_transaction`, `close_batch_transaction` and `close_batch` have been refactored so that the verification harnesses call the functions with an already deserialized proposal (as described in I-05).
    - `execute_transaction`: calls to `ExecutableTransactionMessage::new_validated` and `ExecutableTransactionMessage::execute_message` are ignored by verification.
    - `execute_settings_transaction`: `SettingsAction::AddSpendingLimit` and `SettingsAction::RemoveSpendingLimit` are ignored by verification.
    - `execute_batch_transaction`, `sync_transaction`, `sync_settings_transaction` : same assumptions as `execute_transaction`.

- `create_transaction`: the conversion from `TransactionMessage` to `SmartAccountTransactionMessage` has been replaced with a nondeterministic `SmartAccountTransactionMessage`.
- `create_smart_account`: ignored by the verification.

Formal Verification Properties

sync_transactions/sync_settings_transactions

P-01. [New] Integrity of `sync_transaction` and `sync_settings_transaction`

Status: Verified

Rule Name	Status	Description	Link to rule report
sync_transaction_integrity	Verified	*If sync_transaction succeeds, then the time_lock must be 0, num_signers >= threshold and the sync_transcation signers are also the settings signers.*	[Report]()
sync_settings_transaction_integrity	Verified	*If sync_settings_transaction succeeds, then the time_lock must be 0, num_signers >= threshold and the sync_transcation signers are also the settings signers.*	[Report]()

P-02. [New] No double approve for `sync_transaction` and `sync_settings_transaction`

Status: Verified

Rule Name	Status	Description	Link to rule report

sync_transaction_no_double_approve	Verified	*If sync_transaction succeeds, then there are no duplicate signers in the sync_transcation signers.*	[Report]()
sync_settings_transaction_no_double_approve	Verified	*If sync_settings_transaction succeeds, then there are no duplicate signers in the sync_transcation signers.*	[Report]()

P-03. [New] Equivalence of synchronous transactions and asynchronous transactions

Status: Verified

Rule Name	Status	Description	Link to rule report
sync_tx_succeeds_implies_async_tx_approved	Verified	*If sync_transaction succeeds, then a proposal with the same signers will be approved.*	[Report]()
async_tx_approved_implies_sync_tx_succeeds	Verified	*If approve_proposal succeeds, then sync_transaction with the same signers will succeed.*	[Report]()

P-04. [New] `sync_settings_transaction` preserves `Settings::invariant` and invalidates prior transactions on execution

Status: Verified

Rule Name	Status	Description	Link to rule report
settings_invariant_sync_settings_transaction	Verified	*sync_settings_transaction preserves the Settings::invariant.*	[Report]()

invalidate_prior_transactions_sync_settings_transaction	Verified	*sync_settings_transaction invalidates the prior transactions by updating the stale_transaction_index.*	[Report]()

Allocator

P-05. The function alloc always return valid pointers	
Status: Verified	Prover options: -solanaUsePTA false -useBitVectorTheory true

Rule Name	Status	Description	Link to rule report
rule_integrity_allocator	Verified	*This rule verifies that any pointer returned by alloc is either null or always in-bounds. Moreover, for any given two pointers returned by alloc, they can never alias.*	[Report]()

Settings

P-06. Any function that might modify the settings always calls Settings::invariant and all the invariants described by Settings::invariant hold

Status: Verified	

Rule Name	Status	Description	Link to rule report
settings_invariant_execute_as_authority	Verified	*settings_add_signer, settings_remove_signer,, settings_change_threshold, settings_set_time_lock, settings_new_settings_authority, settings_set_archival_authority*	[Report]()
settings_invariant_tx_create	Verified	*create_transaction, create_settings_transaction, create_batch_transaction*	[Report]()
settings_invariant_settings_tx_execute	Verified	*execute_settings_transaction*	[Report]()

P-07. Any function that might modify the settings consensus parameters always calls invalidate_prior_transactions and settings.transaction_index is always equal to settings.stale_transaction_index

Status: Verified	

Rule Name	Status	Description	Link to rule report
invalidate_prior_transactions_execute_as_authority	Verified	*settings_add_signer, settings_remove_signer,, settings_change_threshold, settings_set_time_lock, settings_new_settings_authority, settings_set_archival_authority*	*Report*
invalidate_prior_transactions_settings_tx_execute	Verified	*execute_settings_transaction*	*Report*

P-08. Integrity of controlled smart account

Status: Verified	

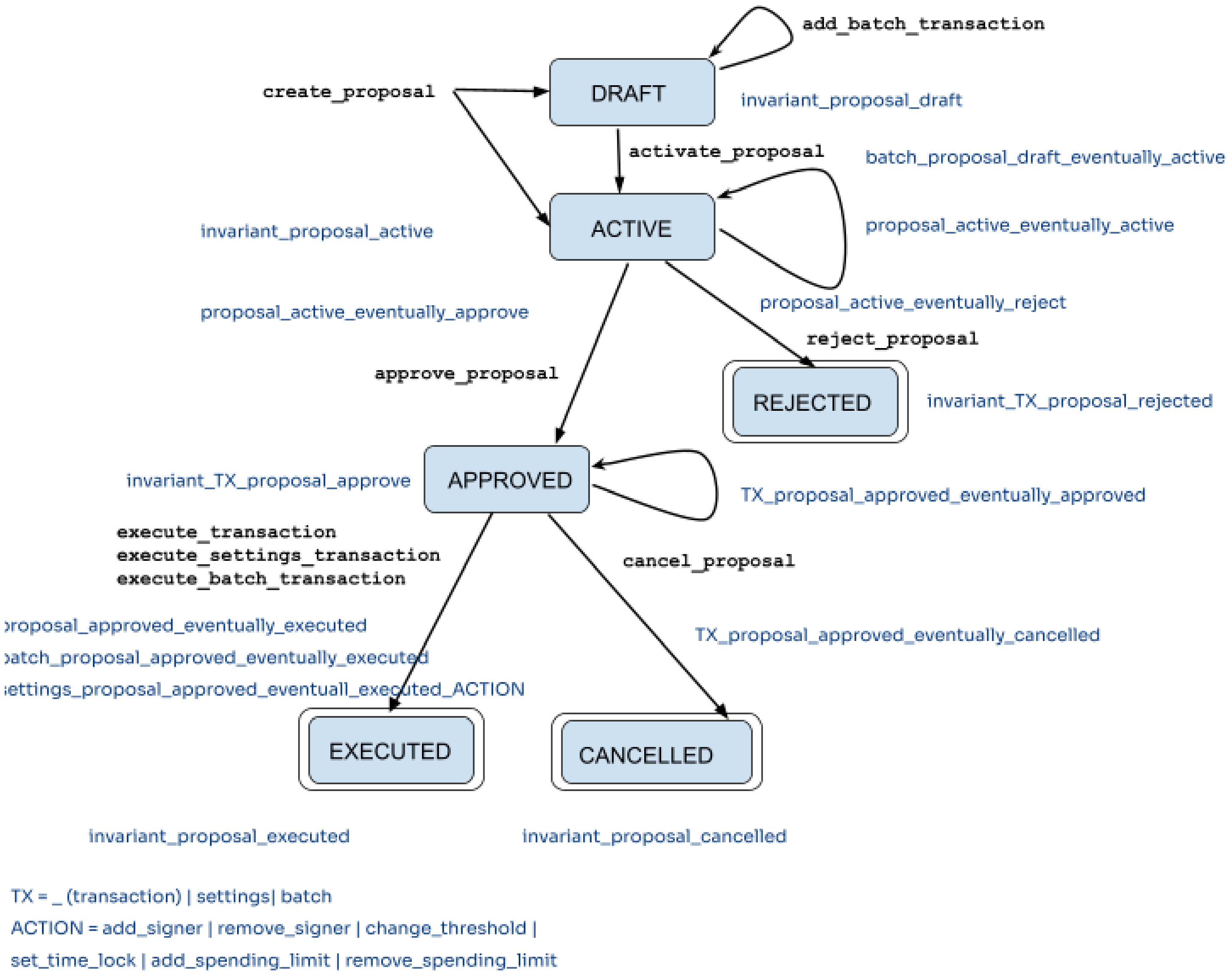
Rule Name	Status	Description	Link to rule report
invariant_no_authority_change	Verified	*Only the settings_authority can call the functions settings_add_signer, settings_remove_signer,, settings_change_threshold, settings_set_time_lock, settings_new_settings_authority, settings_set_archival_authority*	*Report*
integrity_of_settings_add_signer	Verified	*add_signer increases the number of settings signers by 1.*	*Report*
integrity_of_settings_remove_signer	Verified	*Remove_signer decreases the number of settings signers by 1.*	*Report*

P-09. Integrity of non-controlled smart account	
Status: Verified	

Rule Name	Status	Descript ion	Link to rule report
integrity_of_noncontrolled_smart_account	Verified	*The setting.settings_authority must be Pubkey::default()*	[Report]()

Proposal

The following automata shows the different states in which a proposal can be and all its valid transitions. Each state in this automata corresponds to one of the values of ProposalStatus. The rules are in blue. We attach each state and transition to one or more rules. A proposal can be initially either `Draft` or `Active`. While a proposal is active, no settings consensus parameters can be modified, and signers can vote to either approve or reject the proposal. A proposal that is `Rejected` is considered a final state. Once a proposal is `Approved` it can become only either `Executed` or `Canceled`. These two states are also final states.



add_batch_transaction
create_proposal
DRAFT
invariant_proposal_draft
activate_proposal
batch_proposal_draft_eventually_active
invariant_proposal_active
ACTIVE
proposal_active_eventually_active
proposal_active_eventually_approve
proposal_active_eventually_reject
reject_proposal
approve_proposal
REJECTED
invariant_TX_proposal_rejected
invariant_TX_proposal_approve
APPROVED
TX_proposal_approved_eventually_approved
execute_transaction
execute_settings_transaction
execute_batch_transaction
cancel_proposal
roposal_approved_eventually_executed
atch_proposal_approved_eventually_executed
ettings_proposal_approved_eventuall_executed_ACTION
TX_proposal_approved_eventually_cancelled
EXECUTED
CANCELLED
invariant_proposal_executed
invariant_proposal_cancelled
TX = _ (transaction) | settings| batch
ACTION = add_signer | remove_signer | change_threshold |
set_time_lock | add_spending_limit | remove_spending_limit

P-10. The code implements the finite automata depicted above

Status: Verified	

Rule Name	Status	Description	Link to rule report
invariant_proposal_draft	Verified	*If the proposal has status `Draft` then it can only be changed to `Active`*	[Report]()
invariant_proposal_active	Verified	*If the proposal has status `Active` then it can only be changed to `Approved`, `Rejected`, or remains `Active`.* *- If the proposal changed to `Approved` then the size of the approved vector equals the threshold of settings, the size of the rejected vector is less than the cutoff of settings, and the size of the cancelled vector is zero.* *– If the proposal status changed to `Rejected` then the size of the rejected vector is greater than or equal to the cutoff of the settings.* *- If the proposal changed to `Approved` or `Rejected` then the transaction cannot be stale.* *- If the proposal remains `Active` then the size of cancelled is zero.*	[Report]()
proposal_active_eventually_approved	Verified	*(liveness) If the proposal has status `Active` then it can be eventually changed to `Approved`.*	[Report]()

proposal_active_eventually_rejected	Verified	*(liveness) If the proposal has status* `Active` *then it can be eventually changed to* `Rejected`	[Report](#)
proposal_active_eventually_active	Verified	*(liveness) If the proposal has status* `Active` *then it can remain* `Active`*, and the size of approved is less than the threshold, and the size of rejected is less than the cutoff of the settings.*	[Report](#)
invariant_proposal_approved	Verified	*If the proposal has status* `Approved` *then it can only be changed to* `Cancelled`*,* `Executed`*, or remains* `Approved`*.* *- If the proposal changed to* `Executed` *then the time that passed between the proposal was* `Approved` *and its execution is greater or equal than the time_lock of the settings.* *- If the proposal changed to* `Cancelled` *then the size of cancelled is greater than or equal to the settings threshold.* *- The size of approved remains greater than or equal to the threshold of the settings(i.e., the approved vector is not modified even if the proposal is executed or got cancelled)*	
proposal_approved_eventually_executed	Verified	*(liveness) If the proposal has status* `Approved` *then it can be eventually changed to* `Executed`	[Report](#)

proposal_approved_eventually_cancelled	Verified	*(liveness) If the proposal has status `Approved` then it can be eventually changed to `Cancelled`*	[Report]()
proposal_approved_eventually_approved	Verified	*(liveness) If the proposal has status `Approved` then it can remain `Approved`.*	[Report]()
invariant_proposal_rejected	Verified	*If the proposal has status `Rejected` then the proposal status will not change anymore (final state). Moreover, the sizes of approved, rejected, and cancelled vectors do not change.*	[Report]()
invariant_proposal_cancelled	Verified	*If the proposal has status `Cancelled` then the proposal status will not change anymore (final state)*	[Report]()
invariant_proposal_executed	Verified	*If the proposal has status `Executed` then the proposal status will not change anymore (final state)*	[Report]()
invariant_settings_proposal_approved	Verified	*If the settings proposal has status `Approve` then it can only be changed to `Cancelled`, `Executed`, or remains `Approved` - If the proposal changed to `Executed` then the time that passed between the proposal was `Approve` until it was executed is greater or equal than the time_lock of the settings. - If the proposal changed to `Executed` then the transaction cannot be stale.*	[Report]()

		- If the proposal changed to `Cancelled` then the size of cancelled is greater than or equal to the threshold of the settings. *- The size of approved remains greater than or equal to the threshold of the settings (i.e., the approved vector is not modified even if the proposal is executed or got cancelled)*	
settings_proposal_approved_eventually_canceled	Verified	*(liveness) If the settings proposal has status `Approved` then the proposal status can be eventually changed to `Cancelled`.*	[Report]()
settings_proposal_approved_eventually_approved	Verified	*(liveness) If the settings proposal has status `Approved` then the proposal status can remain as `Approved`.*	[Report]()
settings_proposal_approved_eventually_executed_add_signer	Verified	*(liveness) If the settings proposal has status `Approved` then the proposal status can be eventually changed to `Executed`, and the last executed action is `SettingsAction::AddSigner`.*	[Report]()
settings_proposal_approved_eventually_executed_remove_signer	Verified	*(liveness) If the settings proposal has status `Approved` then the proposal status can be eventually changed to `Executed`, and the last executed action is `SettingsAction::RemoveSigner`.*	[Report]()
settings_proposal_approved_eventually_executed_set_time_lock	Verified	*(liveness) If the settings proposal has status `Approved` then the proposal status can be eventually*	[Report]()

		changed to `Executed`, and the last executed action is `SettingsAction::SetTimeLock`.	
settings_proposal_approved_eventually_executed_change_threshold	Verified	*(liveness) If the settings proposal has status `Approved` then the proposal status can be eventually changed to `Executed`, and the last executed action is `SettingsAction::ChangeThreshold`.*	[Report]()
settings_proposal_approved_eventually_executed_add_spending_limit	Verified	*(liveness) If the settings proposal has status `Approved` then the proposal status can be eventually changed to `Executed`, and the last executed action is `SettingsAction::AddSpendingLimit`.*	[Report]()
settings_proposal_approved_eventually_executed_remove_spending_limit	Verified	*(liveness) If the settings proposal has status `Approved` then the proposal status can be eventually changed to `Executed`, and the last executed action is `SettingsAction::RemoveSpendingLimit`.*	[Report]()
batch_proposal_draft_eventually_active	Verified	*(liveness) If a batch transaction has status `Draft` can be eventually changed to `Active`*	[Report]()
invariant_batch_proposal_approve	Verified	*If the batch proposal has status `Approved` then it can only be changed to `Cancelled`, `Executed`, or remains `Approved`.* - *If the proposal changed to `Executed` then the*	[Report]()

		time that passed between the proposal was `Approved` until it was executed is greater than or equal to the time_lock of the settings. - If the proposal changed to `Executed` then the size of the batch is equal to executed_transaction_index - If the proposal changed to `Approved` then the size of the batch is greater than executed_transaction_index - If the proposal changed to `Cancelled` then the size of cancelled is greater than or equal to the threshold of the settings. - The size of approved remains greater than or equal to the threshold of the settings (i.e., the approved vector is not modified even if the proposal is executed or got cancelled)	
batch_proposal_approved_eventually_executed_1 **batch_proposal_approved_eventually_executed_2**	Verified	*(liveness) If a batch transaction has status `Approved` can be eventually changed to `Executed`*	Report Report
batch_proposal_approved_eventually_cancelled	Verified	*(liveness) If a batch transaction has status `Approved` can be eventually changed to `Cancelled`*	Report

batch_proposal_approved_eventually_approved	Verified	*(liveness) If a batch transaction has status `Approved` can remain in `Approved`*	[Report]()

P-11. Proposal has always enough allocated space	
Status: Verified	

Rule Name	Status	Description	Link to rule report
proposal_has_enough_allocated_space_1 **proposal_has_enough_allocated_space_2**	Verified	*The number of bytes occupied by a proposal is less or equal than the actual allocated space for the proposal which must be always bounded by the current number of settings signer.*	[Report]() [Report]()

P-12. Proposal: No double approve/cancel/reject	
Status: Verified	

Rule Name	Status	Description	Link to rule report
proposal_no_double_approve	Verified	*The same signer cannot approve twice the same active proposal*	[Report]()

proposal_no_double_reject	Verified	*The same signer cannot reject twice the same active proposal*	[Report]()
proposal_no_double_cancel	Verified	*The same signer cannot cancel twice the same approved proposal*	[Report]()

P-13. Proposal becomes stale if settings consensus parameter changes

Status: Verified

Rule Name	Status	Description	Link to rule report
proposal_stales_if_settings_changes	Verified	*If a proposal is in an arbitrary state and then if either , settings_add_signer, settings_remove_signer,, settings_change_threshold, settings_set_time_lock or settings_new_settings_authority is executed then the proposal becomes stale*	[Report]()

close_account

P-14. Integrity of close account	
Status: Verified	

Rule Name	Status	Description	Link to rule report
integrity_of_close_transaction	Verified	*close_transaction can only succeed if the proposal is in a final state (Rejected, Cancelled, or Executed) or is stale but not Approved*	Report
integrity_of_close_transaction_no_proposal	Verified	*If a transaction does not have a proposal then close_transaction can only succeed if the transaction is stale.*	Report
integrity_of_close_settings_transaction	Verified	*close_settings_transaction can only succeed if the proposal is in final state (Rejected, Cancelled, or Executed) or is stale.*	Report
integrity_of_close_settings_transaction_no_proposal	Verified	*If a settings transaction does not have a proposal then close_settings_transaction can only succeed if the transaction is stale.*	Report
integrity_of_close_batch_1 **integrity_of_close_batch_2**	Verified	*If the function close_batch_accounts does not revert then all its vault batch transactions have been previously closed (i.e., size of the batch is 0) and the proposal is either Executed, Rejected or Cancelled or if it is stale then it cannot be Approved.*	Report 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.