Multi-Gov: Cross Chain Governance Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

0kage

February 18, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

MultiGov is a cross-chain governance system designed to enable decentralized decision-making across multiple blockchain networks. The protocol allows token holders to participate in governance activities, such as creating proposals and voting, regardless of the specific chain on which their tokens reside.

Key components of the MultiGov system include:

Hub chain: Serves as the central coordination point for cross-chain governance activities. It hosts the main governance contract (HubGovernor) and manages the aggregation of votes from different chains.

Spoke chains: Represent the various blockchain networks integrated into the MultiGov system. Each spoke chain contains contracts that facilitate local voting and communication with the hub.

Cross-Chain communication: Utilizes the Wormhole protocol to enable secure message passing between the hub and spoke chains.

Fractional Voting: Supports a sophisticated voting mechanism where users can split their voting power across different options (For, Against, Abstain).

Vote aggregation: Implements a system to collect and combine votes from multiple chains, ensuring fair representation of all token holders in the governance process.

Extensibility: Includes features for extending proposal deadlines to account for disruption in cross-chain messaging.

The protocol aims to solve the challenge of fragmented governance in multi-chain ecosystems, providing a unified decision-making platform for decentralized communities spread across different blockchain networks.

5 Audit Scope

Cyfrin conducted a security audit of the MultiGov - Crosschain Governance system, focusing specifically on the EVM-compatible codebase. The audit was performed on the codebase at commit hash d899f66. This is a follow-up to a previous audit of the codebase performed by Cyfrin at commit hash 1361e60.

The audit focused on identifying potential security vulnerabilities, logical errors, and adherence to best practices within these smart contracts. Special attention was given to the cross-chain communication and vote decoding logic from the Solana (spoke) to Ethereum (hub). This audit was limited only to the Solidity smart contracts and did not include any non-EVM contracts, off-chain components or front-end applications.

Following contracts in the evm folder were included in the scope of the audit:

```
    src/HubSolanaMessageDispatcher.sol
    src/HubSolanaSpokeVoteDecoder.sol
    src/HubEvmSpokeAggregateProposer.sol
    src/HubEvmSpokeVoteDecoder.sol
    src/HubGovernor.sol
    src/HubMessageDispatcher.sol
    src/HubProposalExtender.sol
    src/HubProposalMetadata.sol
    src/HubVotePool.sol
    src/SpokeAirlock.sol
    src/SpokeMessageExecutor.sol
    src/SpokeMetadataCollector.sol
    src/SpokeVoteAggregator.sol
    src/WormholeDispatcher.sol
    src/extensions/GovernorMinimumWeightedVoteWindow.sol
    src/extensions/GovernorSettableFixedQuorum.sol
    src/lib/Checkpoints.sol
    src/lib/GovernorCountingFractional.sol
    src/lib/SpokeCountingFractional.sol
    src/interfaces/IHubVoteExtender.sol
    src/interfaces/ISpokeVoteDecoder.sol
    src/interfaces/IVoteExtender.sol
    script/DeployHubContractsBaseImpl.s.sol
    script/DeployHubContractsEthDevnet1.sol
    script/DeployHubContractsHolesky.sol
    script/DeployHubContractsSepolia.sol
    script/DeploySpokeContractsBaseImpl.sol
    script/DeploySpokeContractsOptimismSepolia.sol
    script/DeploySpokeContractsEthDevnet2.sol
    script/DeploySpokesonHubTestnet.s.sol
```

6 Executive Summary

Over the course of 16 days, the Cyfrin team conducted an audit on the Multi-Gov: Cross Chain Governance smart contracts provided by Wormhole Foundation. In this period, a total of 4 issues were found.

The Wormhole Cross-Chain Voting system enables decentralized governance across multiple blockchains by allowing token holders to participate in voting without requiring token transfers between chains. This mechanism aggregates voting power across different chains while maintaining the integrity and security of the governance process.

The audit focused on EVM implementation of the cross-chain voting system, including vote aggregation, weight calculation, and cross-chain message verification components. The audit revealed a robust codebase with no critical, high, medium, or low-risk vulnerabilities.

The codebase demonstrates a strong security-first approach with multiple layers of validation that include:

- Comprehensive signature verification for cross-chain messages
- Strict validation of vote weights and calculations
- Robust checks for proposal states and voting eligibility
- Thorough validation of cross-chain message formats and data integrity

The implementation includes comprehensive fuzz testing covering various edge cases and scenarios, particularly in vote weight calculations and cross-chain message verification.

The audit identified only minor improvements to the existing codebase that do not impact the core functionality or security of the system. All the issues reported during the audit were resolved by the Wormhole team.

Overall, the Wormhole cross chain voting implementation exhibits exceptional quality with strong emphasis on security.

Summary

Project Name	Multi-Gov: Cross Chain Governance
Repository	multigov
Commit	d899f66a7d2c...
Audit Timeline	Jan 20th - Feb 10th
Methods	Manual Review, Stateful Fuzzing

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	0
Informational	2
Gas Optimizations	2
Total Issues	4

Summary of Findings

[I-1] Unnecessary exposure of default private keys in deployment scripts	Resolved
[I-2] Inconsistent token decimal configuration across deployment scripts	Resolved
[G-1] Unnecessary `else` keyword in `_scale` function	Resolved
[G-2] Add `unchecked` block to `_reverse` function byte order operations	Resolved

7 Findings

7.1 Informational

7.1.1 Unnecessary exposure of default private keys in deployment scripts

Description: The deployment scripts contain hardcoded private keys with comments warning against production use. Although validation logic ensures these default keys can never be used, embedding private keys in source code - even if non-functional - goes against security best practices and should be avoided.

This pattern is found in multiple deployment scripts:

```
// RegisterSpokesOnHubTestnet.sol
uint256 constant DEFAULT_DEPLOYER_PRIVATE_KEY =
    uint256(0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80);

// DeploySpokeContractsBaseImpl.sol
uint256 constant DEFAULT_DEPLOYER_PRIVATE_KEY =
    uint256(0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80);

  // DeployHubContractsBaseImpl.s.sol
  uint256 constant DEFAULT_DEPLOYER_PRIVATE_KEY =
    uint256(0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80);
```

The current implementation provides this default value but ensures it can never be used through validation in `_deploymentWallet`:

```
  function _deploymentWallet() internal virtual returns (Vm.Wallet memory) {
    uint256 deployerPrivateKey = vm.envOr("DEPLOYER_PRIVATE_KEY", DEFAULT_DEPLOYER_PRIVATE_KEY);

    Vm.Wallet memory wallet = vm.createWallet(deployerPrivateKey);
    if (deployerPrivateKey == DEFAULT_DEPLOYER_PRIVATE_KEY) revert InvalidAddressConfiguration();
    ↪  //@audit reverts here
    return wallet;
  }
```

Since the validation ensures these default keys can never be used, there is no reason to expose them in the source code at all.

Recommended Mitigation: Consider removing the hardcoded private keys and directly require the environment variable to be set. This approach avoids the exposure of default private keys and makes deployment requirements explicit.

```
function _deploymentWallet() internal virtual returns (Vm.Wallet memory) {
    uint256 deployerPrivateKey = vm.envUint("DEPLOYER_PRIVATE_KEY"); // @audit ----> reverts if not set
    return vm.createWallet(deployerPrivateKey);
}
```

Wormhole Foundation: Fixed in PR 248

Cyfrin: Resolved.

7.1.2 Inconsistent token decimal configuration across deployment scripts

Description: The deployment scripts for different networks have inconsistent configurations for `solanaTokenDecimals`. This parameter is used in the `HubSolanaSpokeVoteDecoder` to handle decimal scaling between Solana and EVM tokens during vote aggregation.

This pattern is found in multiple deployment scripts:

```
// DeployHubContractsSepolia.sol
solanaTokenDecimals: 6

// DeployHubContractsHolesky.sol
solanaTokenDecimals: 8

// DeployHubContractsEthDevnet1.sol
solanaTokenDecimals: 8
```

The inconsistency in decimal configuration could lead to incorrect vote weight scaling between Solana and Evm chains and also create potential confusion during testnet deployment.

Recommended Mitigation: Consider documenting the reason for different decimal configuration, if intentional. If not intentional, consider standardizing decimals across all deployment scripts.

Wormhole Foundation: Fixed in PR 247.

Cyfrin: Acknowledged.

7.2 Gas Optimization

7.2.1 Unnecessary `else` keyword in `_scale` function

Description: In `HubSolanaSpokeVoteDecoder.sol`, the `_scale` function includes an unnecessary `else` keyword. The logic flow will naturally fall through to the final return statement if none of the previous conditions are met.

Recommended Mitigation: Consider removing the `else` keyword.

Wormhole Foundation: Fixed in PR 238.

Cyfrin: Acknowledged.

7.2.2 Add `unchecked` block to `_reverse` function byte order operations

Description: In `HubSolanaSpokeVoteDecoder.sol`, the `_reverse` function performs byte order reversal through bitwise operations. These operations are mathematically incapable of causing overflow/underflow as they merely rearrange existing bits through shifts and bitwise operations. The current implementation includes unnecessary overflow/underflow checks that consume additional gas.

Recommended Mitigation: Consider wrapping the operations in an unchecked block to avoid unnecessary overflow/underflow checks.

Wormhole Foundation: Fixed in PR 239

Cyfrin: Acknowledged.