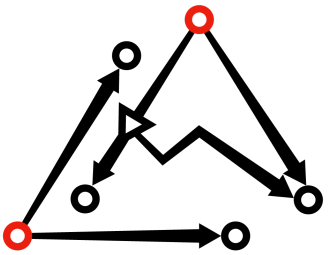

Least Authority

PRIVACY MATTERS

Sygma

Security Audit Report

# Chainsafe

Final Audit Report: 17 May 2023

# Table of Contents

# Overview

## Background

Chainsafe has requested that Least Authority perform a security audit of their Sygma.

## Project Dates

- **February 20 - April 7:** Initial Code Review *(Completed)*
- **April 11:** Delivery of Initial Audit Report *(Completed)*
- **April 20 - May 11:** Verification Review *(Completed)*
- **May 17:** Delivery of Final Audit Report *(Completed)*

## Review Team

- Nicole Ernst, Security Researcher and Engineer
- Sven M. Hallberg, Security Researcher and Engineer
- Steven Jung, Security Researcher and Engineer
- Jan Winkelmann, Cryptography Researcher and Engineer

# Coverage

## Target Code and Revision

For this audit, we performed research, investigation, and review of the Sygma followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repositories are considered in scope for the review:

- Sygma Solidity: https://github.com/sygmaprotocol/sygma-solidity
- Sygma Relayer: https://github.com/sygmaprotocol/sygma-relayer
- Sygma Substrate Pallets: https://github.com/sygmaprotocol/sygma-substrate-pallets

Specifically, we examined the Git revisions for our initial review:

- Sygma Solidity: `bdf0c6692da4151ee020e8151c25ed2aa0417a56`
- Sygma Relayer: `866e68bab8560ad90fc391a38052eabaf9e755c4`
- Sygma Substrate Pallets: `9dddcd77d1dad41ba7012896ae4a180d222da00f`

For the review, these repositories were cloned for use during the audit and for reference in this report:

- Sygma Solidity: https://github.com/LeastAuthority/Chainsafe_Sygma_Solidity
- Sygma Relayer: https://github.com/LeastAuthority/Chainsafe_Sygma_Relayer
- Sygma Substrate Pallets: https://github.com/LeastAuthority/Chainsafe_Sygma_Substrate_Pallets

For the verification, we examined the Git revisions:

- Sygma Solidity: `2396f67292eaa5c5165f05cea23f378a4cd9261b`
- Sygma Relayer: `04fff610b2c061c8bbf821aaadcce48aa36c5bdc`
- Sygma Substrate Pallets: `929b42551f55c9e6c5c34a214ce5747da08f4f68`

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

## Supporting Documentation

The following documentation was available to the review team:

- Sygma:
  https://docs.buildwithsygma.com/
- ChainBridge contract audit:
  https://hackmd.io/@N1kola/HyLDeNfu5
- Previous audit for relayer and contracts:
  Smart Contract and Relayer Security Analysis
- Sygma Relayer SoW for audit:
  https://hackmd.io/@P1sar/H16i9Ytoc

In addition, this audit report references the following documents and links:

- D. Boneh, S. Eskandarian, L. Hanzlik, and N. Greco, "Single Secret Leader Election." *IACR Cryptology ePrint Archive*, 2020, [BEH+20]
- N. J. Smith, "Notes on structured concurrency, or: Go statement considered harmful." 2018, [Smith18]
- M. Sústrik, "Structured Concurrency." 2016, [Sústrik16]
- FRAME macros | Substrate_Docs:
  https://docs.substrate.io/reference/frame-macros/
- NatSpec Format:
  https://docs.soliditylang.org/en/v0.8.19/natspec-format.html

## Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Common and case-specific implementation errors;
- Adversarial actions and other attacks on the bridge;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of service (DoS) attacks and security exploits that would impact or disrupt execution of the bridge;
- Vulnerabilities in the code and whether the interactions between the related and network components are secure;
- Exposure of any critical information during interactions with any external libraries;
- Proper management of encryption and signing keys;
- Protection against malicious attacks and other methods of exploitation;
- Data privacy, data leaking, and information integrity;
- Inappropriate permissions and excess authority; and
- Anything else as identified during the initial analysis phase.

# Findings

## General Comments

Chainsafe Sygma provides an interoperability layer between blockchains and is intended to facilitate cross-chain functionality for dApps. The system is composed of a network of validators that run the relayer implementation, and smart contracts on the source and destination chains. In the current implementation, Sygma includes Ethereum and Substrate smart contract suites to facilitate interoperability between the Ethereum and Substrate ecosystems.

Our team performed a comprehensive review of Chainsafe Sygma's design and implementation, investigating the areas of concern listed above in order to identify vulnerabilities that could affect the security of the system and its users. We investigated the workflow of the relayer, including listening for and parsing chain events, creating and signing transactions, and performing smart contract function calls. We did not identify any vulnerabilities or vectors for state corruption attacks in the relayer.

We also examined the generation of threshold signatures in the relayer, and the respective public keys. Our team noted that the implementation makes use of the third-party library `tss-lib` for this security-critical functionality, which was out of scope for this review. The functionality passing data between the relayer code and the TSS `lib` endpoint is relatively complex due to the concurrent implementation. However, we were unable to find issues that could result in security vulnerabilities, such as data races or incorrect data being passed to or from the third-party library.

Our team also investigated issues related to malformed inputs to the Solidity smart contracts and found that functions either perform sufficient input sanitation, or are only accessible to explicitly permissioned addresses.

### System Design

Our team found that security has been taken into consideration in the design of Chainsafe Sygma. However, during our investigation of the architecture of the system, our team identified some vulnerabilities. The security of the system is reliant on the secure operation of the relayer nodes and the choice of the threshold of relayers needed to successfully execute smart contract function calls.

In our review of the design of the Sygma Substrate pallets, our team did not identify critical security vulnerabilities. We did, however, identify a race condition between expected fees and actual fees that could result in the user paying higher than expected fees (Issue F). Our team noted that the pallets cannot be implemented without support from the (operators of the) target chain since calls are added directly to the chain runtime. In contrast to an EVM smart contract, this means that security issues in the Sygma code can potentially have consequences for the entire chain, rather than being localized to a specific contract or contracts.

The EVM smart contract suite implements security features including role-based permissions, pause functionalities, and token whitelists that improve the security of the system.

### Code Quality

Our team performed a manual review of the three repositories in scope and found that the code is generally well organized. We found that the code style of the relayer component exhibits a pattern of goroutine use without ensuring that the goroutine is finished, and that its resources have been released. We recommend considering using structure concurrency (Suggestion 3).

Our team identified some areas of improvement in the Substrate pallet implementation. We recommend limiting unbounded storage in production code (Suggestion 1), and refactoring access control code to prevent inconsistencies (Suggestion 2). We also recommend that example and "development" code (dev node) be more clearly separated (or documented as such) from production pallets to improve readability.

We also identified opportunities for improvement in the Solidity implementation, including suggestions for adherence to best practice as well as potential gas optimizations.

#### Tests

Our team found that sufficient unit and end-to-end tests have been implemented in all three repositories in scope to check for implementation errors or unexpected behavior that could lead to security vulnerabilities.

### Documentation

The project documentation provided for this review was insufficient and consisted mostly of a detailed and accurate architecture diagram. However, the diagram did not fully cover all the aspects of the system. Project documentation should provide detailed descriptions of each of the system's components, their interaction, and intended functionality.

#### Code Comments

All three repositories include code comments that sufficiently describe the intended behavior of security-critical components and functions. The Solidity implementation code comments adhere to NatSpec format.

### Scope

Our team found that the security of the relayer strongly depends on the security of `chainbridge-core`. As a result, we recommend that this be audited as well.

## Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

| ISSUE / SUGGESTION | STATUS |
|---|---|
| Issue A: Topology Configuration Not Sufficiently Authenticated for Key Refreshes (Known Issue) | Resolved |
| Issue B: Constant Weight Assigned to Linear-complexity Substrate Call | Resolved |
| Issue C: Invalid Proposal Data Could Crash Substrate Runtime | Resolved |
| Issue D: Missing Overflow Check in deposit Function | Resolved |
| Issue E: Missing Overflow Check in execute_proposal Function | Resolved |
| Issue F: Race Condition Between Deposits and Fee Changes | Unresolved |
| Issue G: Return Value From ERC20 Transfer Ignored | Resolved |

| | |
|---|---|
| Suggestion 1: Avoid pallet::without_storage_info in Production Code | Resolved |
| Suggestion 2: Refactor Boilerplate Access Control Code | Resolved |
| Suggestion 3: Consider Using Structured Concurrency in the Relayer | Resolved |
| Suggestion 4: Ensure Relayer Operators Run Their Own Blockchain Nodes | Unresolved |
| Suggestion 5: Distinguish Variable Names | Resolved |
| Suggestion 6: Remove Unnecessary Visibility From Constructors | Resolved |
| Suggestion 7: Remove Unused Variable Names | Resolved |
| Suggestion 8: Use Suitable Mutability | Resolved |
| Suggestion 9: Emit Events on Important Property Updates | Resolved |
| Suggestion 10: Use Immutable Variables for Unchanged Storage Variables | Resolved |
| Suggestion 11: Check interfaceID of ERC1155 Token Address | Resolved |
| Suggestion 12: Use Custom Errors To Save Gas | Resolved |

## Issue A: Topology Configuration Not Sufficiently Authenticated for Key Refreshes (Known Issue)

**Location**

`topology/encryption.go`

`topology/topology.go#L81-L100`

**Synopsis**

The topology configuration specifies which relayers are part of the signing committee. The authenticity and integrity of this data is required for the reliable and secure operation of the bridge.

The Chainsafe team discovered that the version of the code in scope does not adequately protect the data and subsequently resolved the Issue during the audit.

**Impact**

If Multi-Party Computations (MPC) are not performed successfully, depending on the timing of the attack and the configuration of the smart contract, the adversary could be able to submit malicious transactions.

**Preconditions**

The adversary must be able to replace or modify the encrypted topology configuration, replace legitimate relay public keys with their own, and get the relayers to fetch the tampered version instead of the original. If this happens just before an MPC key generation procedure, a malicious key might be submitted to the bridge smart contract.

#### Feasibility

The attack is very complex and requires additional information to be carried out successfully.

#### Remediation

The Chainsafe team resolved this Issue by including the SHA256 hash of the correct topology file in the on-chain events that trigger MPC key generation ceremonies. The relayers then check that the hash contained in the event matches the hash of the encrypted topology configuration data. This achieves authenticity and integrity of the topology, remediating the vulnerability for key refreshes.

#### Verification

Resolved.

### Issue B: Constant Weight Assigned to Linear-complexity Substrate Call

#### Location

[Chainsafe_Sygma_Substrate_Pallets/src/lib.rs#L547]

#### Synopsis

The Substrate runtime call `execute_proposal` takes an arbitrary batch of proposals to execute. The size of this batch is not taken into account in the call's assigned weight.

#### Impact

Maliciously crafted transactions could cause excessive load, degrading network performance.

#### Feasibility

Straightforward.

#### Technical Details

All extrinsics (runtime calls) added by the Sygma Substrate pallets are assigned a fixed weight of 195,000,000. The call `execute_proposal` takes the `Vec<Proposals>` argument, representing a batch to process in a single transaction. In order to verify the MPC signature on this batch, it must be converted from its Rust representation to the original form by the function `construct_ecdsa_signing_proposals_data`. This operation uses time and space linear in the number of proposals and is accessible to anyone by design. A malicious user could thus craft very large transactions (containing invalid proposals) in order to negatively impact the network's performance while paying disproportionately low fees.

#### Remediation

The number of proposals passed to the `execute_proposal` function should be factored into the call's weight calculation. Alternatively, the number of proposals per transaction could be limited, with a check performed before the function `construct_ecdsa_signing_proposals_data` is called.

We recommend benchmarking the performance of all calls and assigning weights based on the results.

#### Status

The Chainsafe team removed the `without_storage_info` macro but noted that without_storage_info is also used on the `access-segregator`, `basic-fee-handler`, and `fee-handler-router` pallets. As a result, the remediation must be applied to these pallets as well, even though they were not explicitly included in the initial audit report.

#### Verification

Resolved.

## Issue C: Invalid Proposal Data Could Crash Substrate Runtime

#### Location

[Chainsafe_Sygma_Substrate_Pallets/bridge/src/lib.rs#L739]

#### Synopsis

The function `extract_deposit_data` will crash the Substrate runtime with a panic if it encounters very large numbers as part of a (signed) proposal.

#### Impact

Operation of the network could be disrupted as nodes fail to advance the network state.

#### Preconditions

In order for this Issue to occur, the offending proposal must have been signed (unwittingly or maliciously) by relayers.

#### Feasibility

Straightforward.

#### Technical Details

The function `execute_proposal` calls the function `extract_deposit_data` to convert the cross-chain representation of a proposed deposit to its Rust equivalent. The former contains the deposit amount and the length of the recipient address as 256-bit quantities. The function attempts to convert these to the smaller Rust data types `u128` and `usize`, respectively. It uses the standard `expect` method to panic if the conversion fails. Substrate runtime calls, however, must never panic to prevent the disruption of the operation of network nodes.

#### Remediation

We recommend that the `expect` method be strictly avoided in Substrate runtime calls. An `Err` result should be returned in case of failure.

#### Status

The Chainsafe team has resolved the Issue.

#### Verification

Resolved.

## Issue D: Missing Overflow Check in deposit Function

#### Location

[Chainsafe_Sygma_Substrate_Pallets/bridge/src/lib.rs#L487]

#### Synopsis

The `deposit` call performs an unguarded increment on `deposit_nonce`.

#### Impact

An overflow would cause the reuse of a previously assigned deposit number or crash the runtime, depending on build settings.

#### Preconditions

The deposit counter ("nonce") would have to advance close to its maximum value of $2^{64} - 1$.

#### Feasibility

Straightforward.

#### Technical Details

Whenever a deposit is made via the `deposit` function, the 64-bit counter `deposit_nonce` is incremented. While the size of the data type practically guarantees that it can never reach its maximum value in this way, good defensive programming practice would still exclude the possibility of overflow by explicitly checking for it.

The Rust addition operator can either wrap around it or panic on overflow, depending on the build profile.

#### Mitigation

The Chainsafe team should refrain from introducing functionality that could cause the deposit counter to jump close to its maximum value.

#### Remediation

We recommend using the standard `checked_add` method in place of the plain addition operator.

#### Status

The Chainsafe team has resolved the Issue.

#### Verification

Resolved.

### Issue E: Missing Overflow Check in execute_proposal Function

#### Location

[Chainsafe_Sygma_Substrate_Pallets/bridge/src/lib.rs#L752]

#### Synopsis

The function `extract_deposit_data` performs an unguarded addition in a consistency check of its argument.

#### Impact

An overflow could crash the runtime, depending on build settings.

#### Preconditions

In order for this Issue to occur, the runtime must have been built with run-time overflow checks enabled, or a crafted proposal must have been signed (unwittingly or maliciously) by relayers.

#### Feasibility

Straightforward.

#### Technical Details

The `extract_deposit_data` function called by `execute_proposal` takes a byte vector as its argument, which should contain deposit data in a prescribed format consisting of 64 bytes of fixed-size data followed by a variable-length recipient address. The function performs a consistency check in the form `data.len() != (64 + recipient_len)`, the right-hand side of which can overflow if the `recipient_len` field of the deposit data contains a sufficiently large number.

#### Mitigation

Relayers should ensure that no malformed proposals are signed. Building the runtime with run-time overflow checks disabled, as is the default in the "release" build profile, would prevent the panic from being triggered.

#### Remediation

We recommend transforming the expression in question to `data.len() - 64 != recipient_len` since `data.len` is ascertained to be greater or equal to 64 at the beginning of the function. Alternatively, we recommend using the standard `checked_add` method in place of the plain addition operator.

#### Status

The Chainsafe team has resolved the Issue.

#### Verification

Resolved.

### Issue F: Race Condition Between Deposits and Fee Changes

#### Location

[Chainsafe_Sygma_Substrate_Pallets/bridge/src/lib.rs#L458]

#### Synopsis

A deposit could be charged with an unexpected fee if a fee change occurs shortly before the deposit is processed.

#### Impact

Users could bear unexpected costs.

#### Feasibility

The impact could occur either by accident or through deliberate manipulation by the bridge operator.

#### Technical Details

The bridge fee is subtracted from a deposit at the time it arrives via the `deposit` function call, based on the amount that is set at that time. After a user views the (expected) fee, a race condition occurs between the processing of their deposit and a concurrent change to the fee.

#### Mitigation

The Chainsafe team should only change bridge fees with long lead times, and after giving ample warnings to users.

#### Remediation

We recommend passing the expected fee as a separate argument to the `deposit` function call. We also recommend refraining from processing the deposit if the actual fee differs from the expectation.

#### Status

The Chainsafe team stated they will most likely implement a mechanism in the next iteration of the pallet to stop the bridge as needed to change a set fee amount to prevent the issue.

#### Verification

Unresolved.

### Issue G: Return Value From ERC20 Transfer Ignored

#### Location

[Chainsafe_Sygma_Solidity/contracts/XC20Safe.sol#L36]

#### Synopsis

This function ignores the return value from the `transfer` function of the ERC20 token. The false return from the `transfer` function is due to the failure of a token transfer.

#### Impact

The transaction can succeed even if a token transfer fails. This could have downstream effects on the accounting of tokens by the bridge.

#### Preconditions

This Issue is possible if the ERC20 token code to mint has a `transfer` function, which returns `false` when the token transfer fails.

#### Mitigation

The ERC20 token to mint should be confirmed when the return value is not dependent on the success of the token transfer.

#### Remediation

We recommend checking the return value and reverting the transaction when it is `false`.

#### Status

The Chainsafe team has resolved the Issue.

#### Verification

Resolved.

## Suggestions

### Suggestion 1: Avoid pallet::without_storage_info in Production Code

#### Location

[Chainsafe_Sygma_Substrate_Pallets/bridge/src/lib.rs#L61]

#### Synopsis

The Substrate attribute macro `#[pallet::without_storage_info]` is used on the bridge pallet. Since it allows blanket unbounded storage for all items in the pallet, it should never be used in a production environment.

#### Mitigation

The `#[pallet::unbounded]` attribute macro should be used instead on those (and only those) storage items that require unbounded size. See also the following document.

#### Status

The Chainsafe team has resolved the suggestion.

#### Verification

Resolved.

### Suggestion 2: Refactor Boilerplate Access Control Code

#### Location

`bridge/src/lib.rs`

`fee-handler-router/src/lib.rs`

`basic-fee-handler/src/lib.rs`

`access-segregator/src/lib.rs`

#### Synopsis

All administrative functions for the Substrate runtime contain the same boilerplate access control code. This could result in inconsistencies.

#### Mitigation

The check for authorized origins should be better abstracted from the call implementations by means of functions or macros. This also applies to a lesser extent to the check that verifies whether the bridge is paused in the function calls `deposit`, `retry`, and `execute_proposal`.

#### Status

This Chainsafe team has resolved the suggestion.

#### Verification

Resolved.

### Suggestion 3: Consider Using Structured Concurrency in the Relayer

#### Location

`Chainsafe_Sygma_Relayer/comm/p2p/libp2p.go#L81-L87`

`Chainsafe_Sygma_Relayer/tss/common/base.go#L119`

#### Synopsis

In the current state, the relayer code liberally creates goroutines whenever a background task is started. The called function is usually passed a context so it can be canceled, but in some instances goroutines

are leaked, leading to leaked memory. It can also make it hard to keep an overview of which goroutines are running.

One approach the industry developed to avoid these problems is structured concurrency. As explained in [Sústrik16] and [Smith18], the core idea is that all goroutines must finish before the function starting them returns.

#### Mitigation

We recommend considering the use of structured concurrency to prevent goroutines from being leaked.

#### Status

The Chainsafe team has resolved the suggestion.

#### Verification

Resolved.

## Suggestion 4: Ensure Relayer Operators Run Their Own Blockchain Nodes

#### Synopsis

The relayer queries the on-chain events from blockchain nodes. The relayer has no means to check that this data is unmodified or that it has received all events. In order to avoid undermining the distribution of trust gained by the threshold signatures, every relayer operator should run their own blockchain node that their relayer uses for querying. This does not necessarily need to be enforced through technical means, but could just be a shared understanding (possibly backed by a contract) that this is the setup expected from parties participating in the consortium.

#### Mitigation

We recommend verifying – through any subset of social, legal, and technical means – that relayer operators run their own blockchain nodes for querying on-chain events.

#### Status

At the time of the verification, the suggested mitigation has not been resolved.

#### Verification

Unresolved.

## Suggestion 5: Distinguish Variable Names

#### Location

`Chainsafe_Sygma_Solidity/contracts/ERC721MinterBurnerPauser.sol#L27`

#### Synopsis

`baseURI` shadows an existing storage variable. The use of identical variable names for global and internal variables can lead to confusion.

#### Mitigation

We recommend using different names for global and internal variables.

**Status**

The Chainsafe team has resolved the suggestion.

**Verification**

Resolved.

## Suggestion 6: Remove Unnecessary Visibility From Constructors

**Location**

Chainsafe_Sygma_Solidity/contracts/Bridge.sol#L107

Chainsafe_Sygma_Solidity/contracts/Forwarder.sol#L28

Chainsafe_Sygma_Solidity/contracts/Migrations.sol#L9

Chainsafe_Sygma_Solidity/contracts/handlers/FeeHandlerRouter.sol#L44

Chainsafe_Sygma_Solidity/contracts/handlers/PermissionedGenericHandler.sol#L47

Chainsafe_Sygma_Solidity/contracts/handlers/PermissionlessGenericHandler.sol#L27

Chainsafe_Sygma_Solidity/contracts/handlers/fee/BasicFeeHandler.sol#L44

Chainsafe_Sygma_Solidity/contracts/utils/AccessControlSegregator.sol#L22

**Synopsis**

The aforementioned constructors have `public` visibility, but these are ignored and do not need to be defined since visibility for constructors has been deprecated.

**Mitigation**

We recommend removing all `public` visibility from constructors.

**Status**

The Chainsafe team has resolved the suggestion.

**Verification**

Resolved.

## Suggestion 7: Remove Unused Variable Names

**Location**

Chainsafe_Sygma_Solidity/contracts/handlers/ERC1155Handler.sol#L37

**Synopsis**

This function defines a named `return` variable. However, it is neither used nor updated, and the function returns empty bytes.

**Mitigation**

We recommend removing the unnecessary name from the `return` variable.

**Status**
The Chainsafe team has resolved the suggestion.

**Verification**
Resolved.

## Suggestion 8: Use Suitable Mutability

**Location**
Chainsafe_Sygma_Solidity/contracts/handlers/ERCHandlerHelpers.sol#L98

Chainsafe_Sygma_Solidity/contracts/handlers/ERCHandlerHelpers.sol#L116

**Synopsis**
The referenced functions do not modify the state of a smart contract. View mutability can be used for these functions so that state modification can be prevented, and mutability be made explicit.

**Mitigation**
We recommend using appropriate mutability for functions.

**Status**
The Chainsafe team has resolved the suggestion.

**Verification**
Resolved.

## Suggestion 9: Emit Events on Important Property Updates

**Location**
Chainsafe_Sygma_Solidity/contracts/handlers/fee/DynamicFeeHandler.sol#L91

Chainsafe_Sygma_Solidity/contracts/handlers/fee/DynamicFeeHandler.sol#L101

**Synopsis**
The referenced functions update important property variables but do not emit corresponding events.

**Mitigation**
We recommend emitting events upon updating important properties.

**Status**
The Chainsafe team has resolved the suggestion.

**Verification**
Resolved.

## Suggestion 10: Use Immutable Variables for Unchanged Storage Variables

**Location**
Chainsafe_Sygma_Solidity/contracts/Migrations.sol#L6

#### Synopsis

This storage variable is set in a constructor but is not intended to be changed later.

#### Mitigation

We recommend using immutable variables for the storage variables that are set only once in a constructor to save gas.

#### Status

The Chainsafe team has resolved the suggestion.

#### Verification

Resolved.

### Suggestion 11: Check interfaceID of ERC1155 Token Address

#### Location

[Chainsafe_Sygma_Solidity/contracts/handlers/ERC1155Handler.sol#L37]

#### Synopsis

This function locks or burns the ERC1155 token, but it does not confirm if the token address is correct. The `interfaceID` was defined already in the contract, and it can be used for checking the token address.

#### Mitigation

We recommend checking the token address using `interfaceID`.

#### Status

The Chainsafe team has resolved the suggestion.

#### Verification

Resolved.

### Suggestion 12: Use Custom Errors To Save Gas

#### Location

Examples (non-exhaustive):

[Chainsafe_Sygma_Solidity/contracts/Bridge.sol#L88]

[Chainsafe_Sygma_Solidity/contracts/ERC20Safe.sol#L101]

#### Synopsis

The smart contracts currently use `require` and `revert` string messages for error handling. However, using a `revert` with a custom `Error`, instead of a string error message, considerably reduces the smart contract size and optimizes gas costs on deployment.

#### Mitigation

We recommend using a `revert` with a custom `Error` to handle errors.

**Status**

The Chainsafe team has implemented the suggestion but noted some exceptions in cases where the current implementation provides better information.

**Verification**

Resolved.

# About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

# Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

## Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

## Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

## Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

## Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

## Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.