Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Sygma

Veridise Inc.

March 22, 2024

► **Prepared For:**

Sygma
https://buildwithsygma.com/

► **Prepared By:**

Shankara Pailoor
Tim Hoffman

► **Contact Us:** contact@veridise.com

► **Version History:**

March. 22, 2024 V1
March. 04, 2024 Draft

Contents

1 Executive Summary

From Jan. 22, 2024 to Feb. 26, 2024, Sygma engaged Veridise to review the security of Spectre, their ZK coprocessor designed to verify block headers from Ethereum's Beacon Chain, as well as sygma-x-solidity, their cross-chain bridge. The review covered the Halo2 circuits in Spectre as well as the smart contracts in sygma-x-solidity. Veridise conducted the assessment over 10 person-weeks, with 2 engineers reviewing code over 5 weeks on commits ba3850e and 609ca8b for the Spectre and sygma-x-solidity repositories respectively. The auditing strategy involved a tool-assisted analysis of the repositories performed by Veridise engineers as well as extensive manual auditing.

Code assessment. The Sygma developers provided the source code of the Spectre circuits and sygma-x-solidity contracts for review. The Spectre circuits were written using Axiom, a popular library wrapper around Halo2, and were designed to provide the same functionality as the circuits from the Telepathy protocol, which were written in Circom. As such, Veridise auditors were able to reuse the documentation from Telepathy's website, along with prior audit reports on the Telepathy code base, as a starting point. While the sygma-x-solidity GitHub repository did not contain detailed README information, Veridise auditors did find relevant documentation about its architecture and design decisions on the Sygma website.

Both repositories contained test suites which exercised the basic functionality of the code bases. In the Spectre repository, the tests mainly ensured the functional correctness of the witness generator and circuits. In particular, the Spectre tests made sure that the witness generator correctly performed public key aggregation and generated proofs that satisfied the constraints. Veridise auditors did not find any tests which checked that the constraints ruled out incorrect witnesses, and recommend the developers add some of these negative test cases.

Summary of issues detected. The audit uncovered 24 issues, 2 of which are assessed to be of high or critical severity by the Veridise auditors. V-CHAIN-VUL-001 details a critical bug in Spectre where the developers forgot to check whether the public keys passed in lie on the BLS12-381 curve. This would permit an attacker to pass in points that sum to a public key they control, allowing them to forge signatures. V-CHAIN-VUL-002 was an issue in Spectre where the developers did not validate that at least two thirds of the validators signed the block header. This issue would allow a malicious validator to forge signatures by setting the number of participants to one when generating the proof. The Veridise auditors also identified 3 medium-severity issues, including V-CHAIN-VUL-004 in sygma-x-solidity where an attacker could apply a second-preimage attack during a Merkle proof to set a non-canonical state root. Veridise auditors also found 8 warnings and 10 informational findings. The Sygma developers acknowledged all the issues and have started generating fixes for them.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve Sygma. In particular, the READMEs of the repositories could be improved to include documentation (or links to documentation) about the code base. Currently, they only describe

how to build each codebase and run the tests. The code base could additionally benefit from adding more test cases in both repositories to increase the code coverage. The test suites should include unit tests for individual functions or components as well as end-to-end tests for both scenarios that are expected to succeed and scenarios that are expected to fail due to access restrictions, etc. Supplementing testing with a fuzzer like afl.rs would be helpful to check the functional correctness of different components like public key aggregation and Poseidon/Sha256 hashing. In this audit, fuzz testing of the Sha256 and Poseidon implementations found four issues described in V-CHAIN-VUL-007, V-CHAIN-VUL-008, V-CHAIN-VUL-017, and V-CHAIN-VUL-024.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise 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.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Spectre	ba3850e	Rust	Halo2
sygma-x-solidity	609ca8b	Solidity	EVM-compatible chains

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Jan. 22 - Feb. 26, 2024	Manual & Tools	2	10 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Fixed	Acknowledged
Critical-Severity Issues	2	2	2
High-Severity Issues	0	0	0
Medium-Severity Issues	3	2	3
Low-Severity Issues	1	1	1
Warning-Severity Issues	8	7	8
Informational-Severity Issues	10	9	10
TOTAL	24	21	24

Table 2.4: Category Breakdown.

Name	Number
Logic Error	9
Data Validation	7
Gas Optimization	5
Maintainability	2
Centralization Risk	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Spectre, Sygma's ZK coprocessor, as well as sygma-x-solidity, which contained the on-chain components of their cross-chain bridge implementation. For each codebase, we sought to answer a distinct set of questions.

For Spectre:

- Does the Spectre protocol allow attackers to forge signatures?
- Can malicious users construct any denial-of-service attacks? In particular, can malicious users prevent benign users from verifying valid proofs?
- Do the circuits validate the Beacon Chain headers according to the validator specifications?
- Do the circuits have any common ZK vulnerabilities such as being underconstrained or overconstrained?

For sygma-x-solidity:

- Does the bridge implementation make sure that users cannot deposit money into a target chain without first locking funds in the source chain?
- Are the on-chain components vulnerable to common vulnerabilities such as re-entrancy attacks?
- Can malicious users perform any denial-of-service attacks on the bridge? In particular, can users prevent any bridge transactions from executing successfully?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool Vanguard. This tool is designed to find instances of common smart contract vulnerabilities, such as re-entrancy and uninitialized variables.
- *Fuzzing/Property-based Testing.* We used afl.rs (a Rust frontend for AFLplusplus) to perform fuzz testing on Spectre's EndianConversions, CommitteeUpdateCircuit, and ssz_merkle implementations. We also performed differential fuzzing on the implementations of the Sha256 and Poseidon hash functions, using pre-existing implementations as oracles.

*Scope.*The scope of the audit consisted of all code in the Spectre and sygma-x-solidity repositories. We note that the Spectre repository implemented its circuits using Axiom, making extensive use of halo2-lib functions. Our audit assumed those functions were implemented correctly.

Methodology. Since the Spectre protocol is intended to copy the functionality of the Telepathy protocol, Veridise auditors reviewed prior audit reports of Telepathy. In particular, any protocol level bugs found in Telepathy were bugs that could be present in the Spectre implementation. For the sygma-x-solidity codebase, Veridise auditors reviewed Sygma's online documentation detailing the overall architecture of the bridge.

They then began a manual audit of the code assisted by both static analyzers and automated testing. During the audit, the Veridise auditors regularly met with the Sygma developers to ask questions about the code. Furthermore, Veridise auditors communicated with Sygma developers over Telegram to discuss issues and fixes between weekly meetings.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-CHAIN-VUL-001	Missing subgroup check on public keys	Critical	Fixed
V-CHAIN-VUL-002	Missing check that the participation sum is gre...	Critical	Fixed
V-CHAIN-VUL-003	Centralization Risk in Bridge	Medium	Acknowledged
V-CHAIN-VUL-004	State Root Verification in Spectre Proxy is Vul...	Medium	Fixed
V-CHAIN-VUL-005	Ineffective concurrency lock	Medium	Fixed
V-CHAIN-VUL-006	Aggregation Logic Could Reject Valid Signatures	Low	Fixed
V-CHAIN-VUL-007	Sha256_wide digest assumes input length to be m...	Warning	Fixed
V-CHAIN-VUL-008	Sha256_wide digest panics when input has more t...	Warning	Fixed
V-CHAIN-VUL-009	Missing Address(0) check in Router and Bridge c...	Warning	Fixed
V-CHAIN-VUL-010	transferHashes map written to but not read from	Warning	Fixed
V-CHAIN-VUL-011	Fee payout susceptible to wasted gas cost	Warning	Acknowledged
V-CHAIN-VUL-012	Missing Address(0) check in renounceAdmin() in ...	Warning	Fixed
V-CHAIN-VUL-013	Using std::env::set_var is unsafe	Warning	Fixed
V-CHAIN-VUL-014	Integer casts may silently truncate	Warning	Fixed
V-CHAIN-VUL-015	X-Coordinate Aggregation could yield point at i...	Info	Fixed
V-CHAIN-VUL-016	Missing Input Validation	Info	Fixed
V-CHAIN-VUL-017	ssz_merkleize_chunks crashes if sync committee ...	Info	Fixed
V-CHAIN-VUL-018	Loss of precision due to floating point log2	Info	Fixed
V-CHAIN-VUL-019	Maintainability Issues	Info	Fixed
V-CHAIN-VUL-020	Gas Optimizations	Info	Partially Fixed
V-CHAIN-VUL-021	Save gas in revert scenarios by using custom er...	Info	Fixed
V-CHAIN-VUL-022	Users can lose funds due to internal decimals c...	Info	Fixed
V-CHAIN-VUL-023	getRoleMemberIndex() result is off by 1	Info	Fixed
V-CHAIN-VUL-024	Handle empty input to poseidon_hash_fq_array	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-CHAIN-VUL-001: Missing subgroup check on public keys

Severity	Critical	**Commit**	ba3850e
Type	Data Validation	**Status**	Fixed
File(s)	`lightclient-circuits/src/committee_update_circuit.rs`		
Location(s)	synthesize()		
Confirmed Fix At	e2a495e		

In the Spectre protocol, a `step` operation is meant to add a new block header from the Beacon chain to the target chain. A key part of this operation is checking that the Ethereum sync committee signed the headers. This is done via zero knowledge proofs.

In particular, the proof needs to do the following steps:

1. Ensure the supplied keys are the same as the sync committee's.
2. Sum the keys to derive an aggregated public key
3. Verify the signature of the headers.

Currently the protocol does not enforce (1). In particular, it does not ensure that the supplied keys are points on the curve let alone the same as the committees. The protocol only ensures that the supplied keys have the same x-coordinates as the sync committee. As a result, a malicious actor has complete freedom in their choice of y-coordinates. As described in the document here, if the attacker has complete freedom in choice of y-coordinates, then they can ensure the aggregated key corresponds to a private key they control. Thus, it would allow them to forge signatures.

Impact Without checking that the public keys are valid points in the subgroup, then attackers could forge signatures.

Recommendation First, we recommend that the Spectre team ensure that the public keys lie on the curve. This can be done by changing `assign_point_unchecked` in `aggregate_pubkeys` to `assign_point`.

However, we don't believe this fix is sufficient because for each `x` coordinate, the attacker still has the freedom to choose `(x, y)` or `(x, -y)` for the corresponding `y` coordinate as well as the participation bit. Under this condition, it is not clear whether the attacker can select participation bits and y-coordinates in such a way that the aggregated sum is a point for which they can recover the corresponding secret key.

The Spectre team suggested updating the rotate circuit to commit each public key's x-coordinate along with its sign. Such a change will ensure that the difficulty of forging signatures is as hard as solving discrete log efficiently. Thus, we support the Spectre team making that change.

Developer Response Fixed in commit `e2a495eb585f45c92fcce48d171aa68c789e8764`

4.1.2 V-CHAIN-VUL-002: Missing check that the participation sum is greater than threshold

Severity	Critical	**Commit**	ba3850e
Type	Logic Error	**Status**	Fixed
File(s)	`contracts/src/Spectre.sol`		
Location(s)	step()		
Confirmed Fix At	a430cae		

The standard way for light clients to check whether to accept a signed header from the Beacon chain is to determine if two thirds of the committee signed it. If no headers have been propagated for a while that meet such a threshold, then clients typically accept the one with the most signatures. At a high level, it is important that clients only accept headers with many signatures because otherwise any malicious committee member (or members) would be able to forge their signature (or collude).

This is because a malicious committee member could set the participation bits to equal 1 for their key and 0 for all other keys else. Thus, the aggregated key would be equal to their public key, and since they have the corresponding secret key, they can forge signatures.

Impact Without any check that the participation sum is above a threshold larger than 1/2, it is feasible for a minority of the committee members to collude and forge signatures that are accepted by Spectre.

Recommendation We recommend that Spectre checks that the participation sum is larger than 2/3's of the committee size.

Developer Response Fixed in commit `a430caeb3678582b73e3ee73b6f001bd9d1e75ca`.

4.1.3 V-CHAIN-VUL-003: Centralization Risk in Bridge

Severity	Medium	**Commit**	609ca8b
Type	Centralization Risk	**Status**	Acknowledged
File(s)	`contracts/utils/AccessControlSegregator.sol`		
Location(s)	adminChangeAccessControl()		
Confirmed Fix At	N/A		

The `AccessControlSegregator` is the module which manages access control for the `Bridge`, `Router`, and `Executor` contracts. Access control is segregated on a per-function basis where for each function, members are assigned roles including admin. One of the functions exposed by the bridge includes the ability to change the segregator itself (see `adminChangeAccessControl`)! A malicious or hacked admin for that function would be able to replace the segregator with a contract they control, thereby compromising the bridge entirely.

Impact A malicious or hacked admin could change the access control module to something they control and gain complete control over the bridge including draining all funds locked in the bridge.

Recommendation We recommend decoupling the ability to change the access control segregator from the segregator itself. One option is to have a governance organization determine what module is set or a multi-sig account that manages the access control.

Developer Response They are using a multi-sig currently. In the future, this will change to a governance contract.

4.1.4 V-CHAIN-VUL-004: State Root Verification in Spectre Proxy is Vulnerable to Preimage Attack

Severity	Medium	**Commit**	609ca8b
Type	Logic Error	**Status**	Fixed
File(s)	`contracts/proxies/SpectreProxy.sol`		
Location(s)	verifyMerkleBranch()		
Confirmed Fix At	6e1d38d		

The Spectre proxy `verifyMerkleBranch()`, shown below, takes as input a state root hash H(denoted by the parameter `leaf`), a merkle proof $P_1, \ldots, P_n$, and the executor root R and verifies the corresponding merkle proof i.e, $Sha256(P_n, Sha256(P_{n-1}, ..., Sha256(P_1, H))) = R$. It uses the state root index to determine the order of the hashing i.e, the path in the tree.

```
function verifyMerkleBranch(
    bytes32 leaf,
    bytes32 root,
    bytes[] calldata proof,
    uint8 index
) internal pure returns (bool) {
    bytes32 value = leaf;

    for (uint256 i = 0; i < proof.length; i++) {
        if ((index / (2**i)) % 2 == 1) {
            value = sha256(abi.encodePacked(proof[i], value));
        } else {
            value = sha256(abi.encodePacked(value, proof[i]));
        }
    }

    return value == root;
}
```

Snippet 4.1: Definition of `verifyMerkleBranch()`

However, the code snippet does not perform any checks on the height of the tree. As such, this snippet is vulnerable to a second pre-image attack whereby an attacker could pass a state root that corresponds to an intermediate node along with a shortened proof.

The only usage of this function is in `step` where the `leaf` node is passed in by the end user.

Impact An attacker could set the state root to be an intermediate node or another leaf in the tree (if the tree is not balanced).

Recommendation We recommend performing a check on the length of the proof.

Developer Response This is fixed in `https://github.com/sygmaprotocol/sygma-x-solidity/pull/37`

4.1.5 V-CHAIN-VUL-005: Ineffective concurrency lock

Severity	Medium	**Commit**	ba3850e
Type	Logic Error	**Status**	Fixed
File(s)	`prover/src/rpc.rs`		
Location(s)	multiple		
Confirmed Fix At	`https://github.com/ChainSafe/Spectre/pull/70`		

The prover sets up an RPC server with 2 handler functions, `gen_evm_proof_committee_update_handler` and `gen_evm_proof_sync_step_compressed_handler`. Its state contains a `tokio::sync::Semaphore` instance to limit the number of concurrent handler executions. The `Semaphore.acquire_owned()` function returns a `Result<OwnedSemaphorePermit, AcquireError>` instance. If the permit is successfully acquired, it is held for the lifetime of the wrapped `OwnedSemaphorePermit` instance.

In both handler functions, the `OwnedSemaphorePermit` is not stored to a local variable within the function so its lifetime ends immediately and the permit is returned to the `Semaphore` before executing the remainder of the function.

```
if let Err(e) = state.concurrency.clone().acquire_owned().await {
  return Err(JsonRpcError::internal(format!(
    "Failed to acquire concurrency lock: {}",
    e
  )));
};
```

Snippet 4.2: Excerpt from `gen_evm_proof_committee_update_handler`

Impact The bodies of `gen_evm_proof_committee_update_handler` and `gen_evm_proof_sync_step_compressed_hand` can be executed by an arbitrary number of threads concurrently. Without a limit, an attacker could execute a DDoS attack on the RPC server.

Recommendation Utilize a structure like the following to perform all actions that must be done while the permit is held:

```
match state.concurrency.clone().acquire_owned().await {
    Err(e) => {
        return Err(JsonRpcError::internal(format!(
            "Failed to acquire concurrency lock: {}",
            e
        )))
    }
    Ok(_permit) => {
        // The permit is held in '_permit' for the scope of this block
    }
}
```

Snippet 4.3: Recommended code structure

Add tests to with concurrent executions to cover this code.

Developer Response Added a local variable to hold the permit until the end of the function scope.

4.1.6 V-CHAIN-VUL-006: Aggregation Logic Could Reject Valid Signatures

Severity	Low	**Commit**	ba3850e
Type	Data Validation	**Status**	Fixed
File(s)	lightclient-circuits/src/sync_step_circuit.rs		
Location(s)	aggregate_pubkeys()		
Confirmed Fix At	2affa66		

The `sync_step_circuit` aggregates the committee public keys using the following logic:

```
let rand_point = g1_chip.load_random_point::<G1Affine>(ctx);
let mut acc = rand_point.clone();
for (bit, point) in participation_bits
            .iter()
            .copied()
            .zip(assigned_affines.iter_mut())
{
    let sum = g1_chip.add_unequal(ctx, acc.clone(), point.clone(), true);
    acc = g1_chip.select(ctx, sum, acc, bit);
}
let agg_pubkey = g1_chip.sub_unequal(ctx, acc, rand_point, false);
```

Snippet 4.4: Excerpt from `sync_step_circuit()`

The sum is computed using the function `add_unequal` which computes the sum of `acc` and `point` and asserts that `acc != point`. However, it is not clear that it is sound to include this assertion since it could be the case that `acc` equals `point` for a valid set of keys. In particular, if we have committee keys with x-coordinates `aG`, `bG` and `(a+b)G`, then this logic would not be able to perform the aggregation on those keys.

Impact This could reject valid committee signatures.

Recommendation We recommend performing a check if the keys are equal and to perform doubling in that case.

Developer Response This is fixed in `https://github.com/ChainSafe/Spectre/pull/63`.

4.1.7 V-CHAIN-VUL-007: Sha256_wide digest assumes input length to be multiple of 4

Severity	Warning	**Commit**	ba3850e
Type	Data Validation	**Status**	Fixed
File(s)	`lightclient-circuits/src/gadget/crypto/sha256_wide.rs`		
Location(s)	digest()		
Confirmed Fix At	108cfc3		

The `Sha256ChipWide` struct implements the `digest()` function to compute the SHA-256 hash of the input data. If the length of the input given to the function is not a multiple of 4, the `digest()` function will panic at the expression `assigned_bytes[i..i + 4]` with the message `"range end index X out of range for slice of length Y"`. The `assigned_bytes` vector is computed from the `input` iterator and has the same length. The line that causes the panic is marked in the following code snippet:

```
for r in 0..num_input_rounds {
   for w in 0..(num_input_words - r * NUM_WORDS_TO_ABSORB) {
      let i = (r * NUM_WORDS_TO_ABSORB + w) * 4;
      let checksum = gate.inner_product(
          builder.main(),
          assigned_bytes[i..i + 4].to_vec(), <---- panic here
          byte_bases.clone(),
      );
      builder
          .main()
          .constrain_equal(&checksum, &blocks[r].word_values[w]);
      }
}
```

Snippet 4.5: Excerpt from `Sha256ChipWide::digest()`

Impact Function will panic with an unhelpful error message.

Recommendation We recommend padding the input to be a multiple of 4 with zeros as is standard.

Developer Response Applied the recommended fix.

4.1.8 V-CHAIN-VUL-008: Sha256_wide digest panics when input has more than 64 elements

Severity	Warning	**Commit**	ba3850e
Type	Data Validation	**Status**	Fixed
File(s)	`lightclient-circuits/src/gadget/crypto/sha256_wide.rs`		
Location(s)	digest()		
Confirmed Fix At	108cfc3		

The `Sha256ChipWide` struct implements the `digest()` function to compute the SHA-256 hash of the input data. If the length of the input given to the function is greater than 64, the `digest()` function will panic at the expression `blocks[r].word_values[w]` with the message `"index out of bounds"`. The line that causes the panic is marked in the following code snippet:

```
for r in 0..num_input_rounds {
   for w in 0..(num_input_words - r * NUM_WORDS_TO_ABSORB) {
      let i = (r * NUM_WORDS_TO_ABSORB + w) * 4;
      let checksum = gate.inner_product(
          builder.main(),
          assigned_bytes[i..i + 4].to_vec(),
          byte_bases.clone(),
      );
      builder
          .main()
          .constrain_equal(&checksum, &blocks[r].word_values[w]); <---- panic here
      }
}
```

Snippet 4.6: Excerpt from `Sha256ChipWide::digest()`

Impact Function will panic with an unhelpful error message.

Recommendation Change the inner loop to iterate over `0..min(NUM_WORDS_TO_ABSORB, num_input_words - r*NUM_WORDS_TO_ABSORB)`

Developer Response Applied the recommended fix.

4.1.9 V-CHAIN-VUL-009: Missing Address(0) check in Router and Bridge constructors

Severity	Warning	**Commit**	609ca8b
Type	Data Validation	**Status**	Fixed
File(s)	`contracts/Router.sol, contracts/Bridge.sol`		
Location(s)	constructor()		
Confirmed Fix At	e6b5cf8		

`Router.sol` and `Bridge.sol` 's constructors takes as input an `accessControl` address which should correspond to access control contract. However, there is no validation that the address has been set properly; if the deployer inadvertently passes in a 0 for the `accessControl` address, then the contract will be rendered unusable.

As such, we recommend adding a check in the constructor to ensure the addresses are not 0.

Impact If deployer passes in an address of value 0 for the `accessControl` address, then the deployed contract will be unusable.

Recommendation Add checks in constructors ensuring the addresses are not 0.

Developer Response This is fixed in commit `https://github.com/sygmaprotocol/sygma-x-solidity/pull/39`.

4.1.10 V-CHAIN-VUL-010: transferHashes map written to but not read from

Severity	Warning	**Commit**	609ca8b
Type	Gas Optimization	**Status**	Fixed
File(s)	`contracts/Router.sol`		
Location(s)	deposit()		
Confirmed Fix At	6c3233c		

The `transferHashes` map keeps track of all processed deposits and is written to in the `deposit` function. However, it is not clear why this map is used since it is only ever written to but not read from. If this map is not used elsewhere, then maintaining it will waste gas for the end users. However, if it should be used elsewhere, then it is a bug.

Impact Excessive gas usage.

Recommendation Add a comment describing why this map is being maintained and who is expected to use it.

Developer Response Added comment explaining usage.

4.1.11 V-CHAIN-VUL-011: Fee payout susceptible to wasted gas cost

Severity	Warning	**Commit**	609ca8b
Type	Gas Optimization	**Status**	Acknowledged
File(s)	multiple		
Location(s)	multiple		
Confirmed Fix At	N/A		

The admin account distributes previously collected fees via the `BasicFeeHandler.transferFee()` and `PercentageERC20FeeHandlerEVM.transferERC20Fee()` functions by passing in an array of fee recipient address and an array of fee amounts. In both functions, for each pair of recipient and fee amount, the relevant funds are transferred and a `FeeDistributed` event is emitted.

The `BasicFeeHandler` performs the transfer by using the low-level `call` function on the recipient address.

```
for (uint256 i = 0; i < addrs.length; i++) {
    (bool success, ) = addrs[i].call{value: amounts[i]}("");
    require(success, "Fee ether transfer failed");
    emit FeeDistributed(address(0), addrs[i], amounts[i]);
}
```

Snippet 4.7: Excerpt from `BasicFeeHandler.transferFee()`

The `PercentageERC20FeeHandlerEVM` performs the transfer via `ERC20Safe.releaseERC20()` which will ultimately call `IERC20.transfer()` on the `tokenAddress` with the relevant parameters.

```
for (uint256 i = 0; i < addrs.length; i++) {
    releaseERC20(tokenAddress, addrs[i], amounts[i]);
    emit FeeDistributed(tokenAddress, addrs[i], amounts[i]);
}
```

Snippet 4.8: Excerpt from `PercentageERC20FeeHandlerEVM.transferERC20Fee()`

In both cases, these transfers are performed in a loop within a single transaction. That means, if any individual transfer call runs out of gas or otherwise causes the transaction to revert, all transfers will be reverted. Similarly, if the admin simply attempts to initiate too many transfers at once, the transaction may run out of gas and revert. When the transaction reverts, all gas cost already incurred while processing the transaction is lost and will not be refunded to the admin.

If the fee recipient address is a smart contract, it can simply run out of gas due to complex operations or revert due to coding mistakes but there is also opportunity for a malicious contract to intentionally revert in order to cause the admin account to waste gas.

Impact This approach requires the admin account to pay the transaction gas cost when performing payouts and performing multiple payouts in a single transaction increases the likelihood of the transaction reverting and the admin losing funds due to wasted gas costs.

Recommendation Perform payments in separate transactions or refactor payouts to a "pull" system instead of a "push" system where recipients are required to initiate fee payouts.

Developer Response Admin is expected to properly validate data off-chain before making the call on-chain.

4.1.12 V-CHAIN-VUL-012: Missing Address(0) check in renounceAdmin() in BasicFeeHandler

Severity	Warning	**Commit**	609ca8b
Type	Data Validation	**Status**	Fixed
File(s)	`contracts/handlers/fee/BasicFeeHandler.sol`		
Location(s)	renounceAdmin()		
Confirmed Fix At	N/A		

The `BasicFeeHandler` contract exposes a function called `renounceAdmin` where an existing admin can renounce themselves as an admin and set another address as the admin. Only an existing admin can successfully execute this function. As such, there should be a check on the new address that it is not address 0, otherwise the admin role could be locked permanently if there was only one admin.

Impact If an admin accidentally calls `renounceAdmin` with `newAdmin = 0` then the adminship can be locked.

Recommendation Add a check that `newAdmin != 0`.

Developer Response The recommendation was applied in `https://github.com/sygmaprotocol/sygma-x-solidity/pull/44`.

4.1.13 V-CHAIN-VUL-013: Using std::env::set_var is unsafe

Severity	Warning	**Commit**	ba3850e
Type	Logic Error	**Status**	Fixed
File(s)	multiple		
Location(s)	multiple		
Confirmed Fix At	abd30f2		

Using the `std::env::set_var()` function is unsafe in multi-threaded programs. See https://doc.rust-lang.org/nightly/std/env/fn.set_var.html#safety. The Spectre `prover` module sets up a multi-threaded RPC server that utilizes the `lightclient_circuits` module which contains multiple calls to `std::env::set_var()` at the following locations:

```
fn set_var(&self) {
    set_var(
        "AGG_CONFIG_PARAMS",
        serde_json::to_string(&self.params).unwrap(),
    );
    set_var("LOOKUP_BITS", (self.params.degree - 1).to_string());
}
```

Snippet 4.9: Definition of `AggregationConfigPinning::set_var()`

```
fn set_var(&self) {
    set_var(
        "GATE_CONFIG_PARAMS",
        serde_json::to_string(&self.params).unwrap(),
    );
    set_var("LOOKUP_BITS", (self.params.k - 1).to_string());
}
```

Snippet 4.10: Definition of `Eth2ConfigPinning::set_var()`

```
set_var(
    "AGG_CONFIG_PARAMS",
    serde_json::to_string(&circuit.calculate_params(Some(10))).unwrap(),
);
```

Snippet 4.11: Excerpt from `AggregationCircuit::create_circuit()`

```
set_var(
    "GATE_CONFIG_PARAMS",
    serde_json::to_string(&params).unwrap(),
);
```

Snippet 4.12: Excerpt from `ShaCircuitBuilder::calculate_params()`

Impact Unrecoverable errors may occur when one thread calls `std::env::set_var()` and another thread either writes or reads from the OS environment via some other means.

Recommendation Instead of environment variables, use a configuration file or Rust global variables to store these modifiable configuration values.

Developer Response Refactored so that circuit configuration is serialized/deserialized from a file.

4.1.14 V-CHAIN-VUL-014: Integer casts may silently truncate

Severity	Warning	**Commit**	ba3850e
Type	Logic Error	**Status**	Fixed
File(s)	multiple		
Location(s)	multiple		
Confirmed Fix At	N/A		

Rust provides the `as` operator to cast values to a different type. When performing casts from a larger integer type to a smaller one, the value will *silently* truncate. For example, a cast from `u64` to `u32` will keep the lower 32 bits of the number and drop the rest which will change the value if the original value required more than 32 bits to represent. See: https://doc.rust-lang.org/reference/expressions/operator-expr.html#semantics

The function `Sha256ChipWide::digest` uses the `as` operator to convert from `u32` to `u8`. If the value of `av.value().get_lower_32()` is larger than `u8::MAX`, the higher bits will be dropped, preserving only the lower 8 bits of the value.

```
fn digest(
    &self,
    builder: &mut Self::CircuitBuilder,
    input: impl IntoIterator<Item = QuantumCell<F>>,
) -> Result<Vec<AssignedValue<F>>, Error> {
    let assigned_bytes = input
        .into_iter()
        .map(|cell| match cell {
            QuantumCell::Existing(v) => v,
            QuantumCell::Witness(v) => builder.main().load_witness(v),
            QuantumCell::Constant(v) => builder.main().load_constant(v),
            _ => unreachable!(),
        })
        .collect_vec();
        let binary_input: HashInput<u8> = HashInput::Single(
            assigned_bytes
                .iter()
                .map(|av| av.value().get_lower_32() as u8)
                .collect_vec()
                .into(),
        );
        ...
```

Snippet 4.13: Excerpt from `Sha256ChipWide::digest()`

There are also several locations that use the `as` operator to convert either from or to the `usize` type whose bit-width depends on the target architecture: on a 32 bit target, it is 32 bits and on a 64 bit target, it is 64 bits.

Impact Truncation of values will go unnoticed and may lead to unexpected errors.

Recommendation Use the `TryFrom` trait instead of the `as` operator so truncation of the value will be detected and must be handled appropriately, either by propagating the `Result::Err` or

```rust
fn degree(&self) -> u32 {
    self.params.k as u32
}
```

Snippet 4.14: Definition of `Eth2ConfigPinning::degree()` in `lightclient-circuits/src/util/circuit.rs`

```rust
circuit.num_instance().first().map_or(0, |x| *x as u32)
```

Snippet 4.15: Excerpt from `gen_evm_verifier()` in `prover/src/cli.rs`

```rust
let sync_period = (bootstrap.header.beacon.slot as usize) /
    EPOCHS_PER_SYNC_COMMITTEE_PERIOD;
```

Snippet 4.16: Excerpt from `get_initial_sync_committee_poseidon()` in `test-utils/src/lib.rs`

using a function like `unwrap()` or `expect()` to panic.

- Use `u8::try_from(X).expect("truncated")` instead of `X as u8`
- Use `u32::try_from(X).expect("truncated")` instead of `X as u32`
- Use `usize::try_from(X).expect("truncated")` instead of `X as usize`

Developer Response The recommendation was applied in all cases.

4.1.15 V-CHAIN-VUL-015: X-Coordinate Aggregation could yield point at infinity

Severity	Info	**Commit**	ba3850e
Type	Logic Error	**Status**	Fixed
File(s)	`lightclient-circuits/src/sync_step_circuit.rs`		
Location(s)	aggregate_pubkeys()		
Confirmed Fix At	98a9362		

The function `aggregate_pubkeys()` sums all the x-coordinates of the committee public keys. It does so using the following algorithm:

```
r <- sampleG1Point()
acc <- r
for (key : keys) {
   assert acc != key.x
   acc += key.x
}
acc <- acc - r;
```

Snippet 4.17: Pseudocode of excerpt from `aggregate_pubkeys()`

It initializes the sum to a random point on the curve and then iteratively adds `key.x` to the sum and finally subtracts the random point. The assertion in the beginning prevents the sum reaching to the point at infinity; however, the subtraction at the end could yield the point at infinity if `acc` is `r` for example.

Impact If a bad `r` is sampled, then the resulting aggregated signature could be the point at infinity which would likely result in the signature validation failing.

However, this is a very unlikely scenario.

Recommendation We would recommend having a special case which initializes the accumulator to the first key and then perform the aggregation.

Developer Response The developers acknowledged the issue and fixed it in `https://github.com/ChainSafe/Spectre/pull/65`.

4.1.16 V-CHAIN-VUL-016: Missing Input Validation

Severity	Info	**Commit**	ba3850e
Type	Gas Optimization	**Status**	Fixed
File(s)	`src/Spectre.sol`		
Location(s)	step(), rotate()		
Confirmed Fix At	N/A		

The purpose of the `step` and `rotate` calls is to facilitate validating and storing the block headers of the beacon chain for each attestation period. Once a header has been verified for an attestation period, there should be no reason to verify it again. As such, it only makes sense to perform these operations when the attestation period is the *latest*.

The current implementation does not check whether the headers have already been verified for a given period and so would allow users to potentially waste gas unnecessarily verifying proofs on chain.

The implementation already keeps track of the latest attestation period in a variable called `head` and should first check if the attestation period passed in is equal to head.

Impact By not checking if the headers for a given period have already been verified, this implementation could unnecessarily waste gas

Recommendation We recommend that the implementation check if a header for a given period has already been verified and revert if it has.

Developer Response The developers implemented the recommendation in the following pull request: `https://github.com/ChainSafe/spectre-contracts/pull/3`

4.1.17 V-CHAIN-VUL-017: ssz_merkleize_chunks crashes if sync committee size is not a power of 2

Severity	Info	Commit	ba3850e
Type	Logic Error	Status	Fixed
File(s)	lightclient-circuits/src/ssz_merkle.rs		
Location(s)	ssz_merkleize_chunks()		
Confirmed Fix At	1a1f9c9		

In `ssz_merkleize_chunks()` the vector of chunks is padded with values from `ZERO_HASHES` to ensure the length is even so that hashing them by pairs later in the loop will succeed.

```
for depth in 0..height {
    // Pad to even length using 32 zero bytes assigned as constants.
    let len_even = chunks.len() + chunks.len() % 2;
    let padded_chunks = chunks
        .into_iter()
        .pad_using(len_even, |_| ZERO_HASHES[depth].as_slice().into_constant())
        .collect_vec();
...
}
```

Snippet 4.18: Excerpt from `ssz_merkleize_chunks()`

The `ZERO_HASHES` array contains padding values for the first two levels. At the first level, when processing the input chunks themselves, `[0; 32]` is used as the padding. For the second level, the `ZERO_HASHES` array has the resulting SHA-256 hash for two chunks of `[0; 32]`.

```
pub const ZERO_HASHES: [[u8; 32]; 2] = [
    [0; 32],
    [
        245, 165, 253, 66, 209, 106, 32, 48, 39, 152, 239, 110, 211, 9, 151, 155, 67,
     0, 61, 35,
        32, 217, 240, 232, 234, 152, 49, 169, 39, 89, 251, 75,
    ],
];
```

Snippet 4.19: Definition of `ZERO_HASHES`

If the length of the input `chunks` is greater than 8 and not a power of 2, the `ssz_merkleize_chunks` will attempt to read `ZERO_HASHES[2]` which does not exist resulting in a Rust runtime error.

All uses of `ssz_merkleize_chunks` in Spectre provide 5 chunks except for the use via `CommitteeUpdateCircuit::synthesize` which provides `eth_types::Spec::SYNC_COMMITTEE_SIZE` chunks.

Impact

1. If an implementation of `eth_types::Spec` is used such that its `SYNC_COMMITTEE_SIZE` is not a power of 2, the crash will occur.
2. Also, the precomputed `ZERO_HASHES[1]` values assume a SHA-256 hash is used. If the `HashInstructions` implementation passed to `ssz_merkleize_chunks` is not SHA-256, the end result would be computed incorrectly.

Recommendation

1. Ensure `SYNC_COMMITTEE_SIZE` is always a power of 2 for every implementation of `Spec` or provide a mechanism to compute additional levels of the computed zero hash result on demand.
2. Compute all levels beyond the `[0; 32]` initial level on demand using the given `HashInstructions` implementation.

Developer Response

1. Added an assertion to explicitly check the input size.
2. Made it clear that SHA-256 hash is used to compute the `ZERO_HASHES` constants since the only implementations of `HashInstructions` within scope are `Sha256Chip` and `Sha256ChipWide`.

4.1.18 V-CHAIN-VUL-018: Loss of precision due to floating point log2

Severity	Info	**Commit**	ba3850e
Type	Logic Error	**Status**	Fixed
File(s)	`lightclient-circuits/src/ssz_merkle.rs`		
Location(s)	ssz_merkleize_chunks()		
Confirmed Fix At	f9d3e5f		

To compute the root hash of the Merkle tree, the input chunks are hashed in pairs, reducing the number of chunks by half. The process repeats until there is a single chunk. Thus the number of iterations required to compute the root hash is log2(size). The `ssz_merkleize_chunks` function computes this using the floating point `f64::log2()` function.

```
pub fn ssz_merkleize_chunks<F: Field, CircuitBuilder: CommonCircuitBuilder<F>>(
    builder: &mut CircuitBuilder,
    hasher: &impl HashInstructions<F, CircuitBuilder = CircuitBuilder>,
    chunks: impl IntoIterator<Item = HashInputChunk<QuantumCell<F>>>,
) -> Result<Vec<AssignedValue<F>>, Error> {
    let mut chunks = chunks.into_iter().collect_vec();
    let len_even = chunks.len() + chunks.len() % 2;
    let height = (len_even as f64).log2().ceil() as usize;
    for depth in 0..height {
            ...
}
```

Snippet 4.20: Excerpt from `ssz_merkleize_chunks`

Impact At large values, this floating point calculation suffers from loss of precision and can compute the height as 1 less than what it should be. Thus the loop would execute too few times leaving more than one chunk in the vector and failing the assertion after the loop.

Recommendation Compute height of tree as `chunks.len().next_power_of_two().ilog2()` with a special case for `chunks.len() == 1` to execute the loop 1 time.

Developer Response Applied the recommended fix.

4.1.19 V-CHAIN-VUL-019: Maintainability Issues

Severity	Info	**Commit**	ba3850e and 609ca8b
Type	Maintainability	**Status**	Fixed
File(s)	multiple		
Location(s)	multiple		
Confirmed Fix At	d428089		

1. Unused `import` and `using` statements:

sygma-x-solidity contracts/Bridge.sol

- `import "@openzeppelin/contracts/utils/cryptography/ECDSA.sol";`
- `using ECDSA for bytes32;`

sygma-x-solidity contracts/Executor.sol

- `import "@openzeppelin/contracts/utils/cryptography/ECDSA.sol";`
- `using ECDSA for bytes32;`

sygma-x-solidity contracts/Router.sol

- `import "@openzeppelin/contracts/utils/cryptography/ECDSA.sol";`
- `using ECDSA for bytes32;`

sygma-x-solidity contracts/utils/AccessControl.sol

- `import "@openzeppelin/contracts/utils/Address.sol";`
- `using Address for address;`

2. Unnecessary cast:

sygma-x-solidity contracts/Executor.sol

- `IBridge(_bridge)`

3. Function can be marked with `pure`:

sygma-x-solidity contracts/libraries/StorageProof.sol

- `getStorageValue()`
- `getStorageRoot()`

4. Function should be marked with `private`:

sygma-x-solidity contracts/ERC20Safe.sol

- `_safeTransfer()`

5. Modifier name is misleading because `Router` and `Executor` are also allowed:

sygma-x-solidity contracts/handlers/ERCHandlerHelpers.sol

- `modifier onlyBridge()`

6. Return value unnecessary because it's always true and is unused in caller:

sygma-x-solidity contracts/Executor.sol

- `function verify() returns (bool)`

7. File is not used:

sygma-x-solidity contracts/libraries/RLPWriter.sol

8. Unused dependencies:

Spectre prover/Cargo.toml

- `anstyle = "1.0.0"`
- `futures = "0.3.29"`

9. Unused annotations:

Spectre eth-types/src/lib.rs

- `#![allow(incomplete_features)]` (not used)
- `#![feature(associated_type_bounds)]` (feature not used)
- `#![feature(associated_type_defaults)]` (feature not used)
- `#![feature(generic_const_exprs)]` (feature not used)

Spectre lightclient-circuits/src/gadget/crypto/sha256_flex/compression.rs

- `#[allow(clippy::too_many_arguments)]` (argument count less than the default, 7)

Spectre lightclient-circuits/src/gadget/crypto/sha256_flex/gate.rs

- `#[allow(clippy::type_complexity)]` (function is within the default complexity limit)

Spectre lightclient-circuits/src/lib.rs

- `#![feature(int_roundings)]` (stable since 1.73.0)
- `#![feature(associated_type_bounds)]` (feature not used)
- `#![feature(stmt_expr_attributes)]` (feature not used)
- `#![feature(trait_alias)]` (feature not used)
- `#![feature(generic_arg_infer)]` (feature not used)

Spectre prover/src/args.rs

- `#[allow(clippy::large_enum_variant)]` (`Circuit` is within the default threshold of 200 bytes)

Spectre prover/src/lib.rs

- `#![allow(incomplete_features)]` (not used)
- `#![feature(generic_const_exprs)]` (feature not used)

Spectre prover/src/main.rs

- `#![feature(associated_type_bounds)]` (feature not used)

10. Use of magic constant:

Spectre lightclient-circuits/src/gadget/crypto/sha256_flex.rs

- Use of constant 9 should be documented or replaced with declaration with a meaningful name

Impact Can make the code more complicated than necessary and more difficult to understand.

Recommendation

- Remove unused/unnecessary code.
- Use the most restrictive access modifiers possible.
- Use clear and descriptive names.

Developer Response Recommendations were applied.

4.1.20 V-CHAIN-VUL-020: Gas Optimizations

Severity	Info	**Commit**	609ca8b
Type	Gas Optimization	**Status**	Partially Fixed
File(s)	multiple		
Location(s)	multiple		
Confirmed Fix At	N/A		

1. The `Executor.executeProposals()` function reads from a certain index in the `Proposal[] memory proposals` parameter multiple times in each iteration of the loop. A small savings on gas cost in every iteration of the loop could be achieved by adding `Proposal memory p = proposals[i];` at the start of the loop and replacing all `proposals[i]` with `p` within the remainder of the loop.

```
for (uint256 i = 0; i < proposals.length; i++) {
  if (isProposalExecuted(proposals[i].originDomainID, proposals[i].depositNonce)) {
    continue;
  }
  bytes32 stateRoot;
  bytes32 storageRoot;
  address routerAddress = _originDomainIDToRouter[proposals[i].originDomainID];

  IStateRootStorage stateRootStorage = IStateRootStorage(_securityModels[proposals[i
    ].securityModel]);
  stateRoot = stateRootStorage.getStateRoot(proposals[i].originDomainID, slot);
  storageRoot = StorageProof.getStorageRoot(accountProof, routerAddress, stateRoot);
  address handler = IBridge(_bridge)._resourceIDToHandlerAddress(proposals[i].
    resourceID);
  IHandler depositHandler = IHandler(handler);
  verify(proposals[i], storageRoot);

  usedNonces[proposals[i].originDomainID][proposals[i].depositNonce / 256] |=
    1 << (proposals[i].depositNonce % 256);
  try depositHandler.executeProposal(proposals[i].resourceID, proposals[i].data)
    returns (
    bytes memory handlerResponse
  ) {
    emit ProposalExecution(proposals[i].originDomainID, proposals[i].depositNonce,
    handlerResponse);
  } catch (bytes memory lowLevelData) {
    emit FailedHandlerExecution(lowLevelData, proposals[i].originDomainID, proposals[
    i].depositNonce);
    usedNonces[proposals[i].originDomainID][proposals[i].depositNonce / 256] &=
        ~(1 << (proposals[i].depositNonce % 256));
    continue;
  }
}
```

Snippet 4.21: Excerpt from `Executor.executeProposals()`

1. The `Router.deposit()` function has several conditions that can cause the function to revert. To save on gas lost when these reverts occur, these conditions should appear as near the start of the function as possible. Specifically, the `if (handler == address(0))` could

be moved earlier to avoid executing the `feeHandler.collectFee()` statement when the function reverts.

```
if (destinationDomainID == _domainID) revert DepositToCurrentDomain();
address sender = _msgSender();
IFeeHandler feeHandler = _bridge._feeHandler();
if (address(feeHandler) == address(0)) {
    require(msg.value == 0, "no FeeHandler, msg.value != 0");
} else {
    // Reverts on failure
    feeHandler.collectFee{value: msg.value}(
        sender,
        _domainID,
        destinationDomainID,
        resourceID,
        depositData,
        feeData
    );
}
address handler = _bridge._resourceIDToHandlerAddress(resourceID);
if (handler == address(0)) revert ResourceIDNotMappedToHandler();
```

Snippet 4.22: Excerpt from `Router.deposit()`

1. The `ERCHandlerHelpers` contract defines the `withdraw()` function as required to implement the `IERCHandler` interface. It is used by the admin "to manually withdraw funds from ERC safes." However, the definition here has an empty body. Functions with an empty body should generally be removed or refactored to avoid scenarios that waste gas and may mislead users.

```
function withdraw(bytes memory data) external virtual override {}
```

Snippet 4.23: Definition of `ERCHandlerHelpers.withdraw()`

1. The `PermissionlessGenericHandler` defines the `setResource()` function as required to implement the `IHandler` interface. It is used by the admin to map a specific resource ID to an `IHandler` instance and assign an ERC20 token address to the `IHandler` instance. Hence, this function is not relevant in the context of transactions involving the native Ethereum currency and is defined here with an empty body. Functions with an empty body should generally be removed or refactored to avoid scenarios that waste gas and may mislead users.

```
function setResource(bytes32 resourceID, address contractAddress, bytes calldata args
    ) external onlyBridge {}
```

Snippet 4.24: Definition of `PermissionlessGenericHandler.setResource()`

Impact see above

Recommendation

1. Use a temporary local to avoid indexing the same memory multiple times.
2. All illegal state conditions that may cause a revert should be moved to the start of the function.
3. Remove the function definition from `ERCHandlerHelpers` contract and make the contract abstract.
4. Add a `revert` to the function body since the operation is not relevant to the native currency workflow.

Developer Response The recommendation was applied in cases 1-3. The developers chose not to make the recommended change for case 4.

4.1.21 V-CHAIN-VUL-021: Save gas in revert scenarios by using custom errors

Severity	Info	**Commit**	609ca8b
Type	Gas Optimization	**Status**	Fixed
File(s)	multiple		
Location(s)	multiple		
Confirmed Fix At	N/A		

Using custom errors rather than `require()` and `revert(string)` reduces the gas cost of deploying contracts and of reverting transactions. See: https://soliditylang.org/blog/2021/04/21/custom-errors/

The `revert(string)` function is used at 3 locations in `MerkleTrie.get()`.

There are 72 uses of the `require()` function across the following files:

- src/contracts/ERC20Safe.sol
- src/contracts/Router.sol
- src/contracts/handlers/ERCHandlerHelpers.sol
- src/contracts/handlers/FeeHandlerRouter.sol
- src/contracts/handlers/PermissionlessGenericHandler.sol
- src/contracts/handlers/fee/BasicFeeHandler.sol
- src/contracts/handlers/fee/PercentageERC20FeeHandlerEVM.sol
- src/contracts/libraries/Bytes.sol
- src/contracts/libraries/MerkleTrie.sol
- src/contracts/libraries/RLPReader.sol
- src/contracts/libraries/StorageProof.sol
- src/contracts/proxies/SpectreProxy.sol
- src/contracts/utils/AccessControl.sol
- src/contracts/utils/AccessControlSegregator.sol
- src/contracts/utils/Pausable.sol

Impact Excess gas cost at contract deployment and when reverting transactions.

Recommendation Replace uses of `require()` and `revert(string)` with revert statements that use custom error types.

Developer Response The recommendation was applied in all cases.

4.1.22 V-CHAIN-VUL-022: Users can lose funds due to internal decimals conversion

Severity	Info	**Commit**	609ca8b
Type	Data Validation	**Status**	Fixed
File(s)	`contracts/handlers/ERCHandlerHelpers.sol`		
Location(s)	convertToInternalBalance()		
Confirmed Fix At	N/A		

The `ERCHandlerHelpers` abstract contract contains functions that aid in bridging when one side of the bridge is an ERC20 token. ERC20 tokens may hold their values according to a different number of decimal places but the bridge needs to represent them consistently so it uses 18 decimal places (this is a very common choice since it is the ratio of Ether and Wei). The `convertToInternalBalance()` function converts values from the decimal places used by the source ERC20 token to the 18 decimal places used internally by the bridge. This function is called by `ERC20Handler.deposit()` while packing the data that is ultimately stored in the `Deposit` event to be used by Relayers to complete the bridging.

```
1  function convertToInternalBalance(address tokenAddress, uint256 amount) internal view
       returns (uint256) {
2      Decimals memory decimals = _tokenContractAddressToTokenProperties[tokenAddress].
       decimals;
3      uint256 convertedBalance;
4      if (!decimals.isSet) {
5          return amount;
6      } else if (decimals.externalDecimals >= DEFAULT_DECIMALS) {
7          convertedBalance = amount / (10 ** (decimals.externalDecimals -
       DEFAULT_DECIMALS));
8      } else {
9          convertedBalance = amount * (10 ** (DEFAULT_DECIMALS - decimals.
       externalDecimals));
10     }
11
12     return convertedBalance;
13 }
```

Snippet 4.25: Definition of `ERCHandlerHelpers.convertToInternalBalance()`

The user of the bridge will lose funds if `decimals.externalDecimals >= 18` and `amount < (10 ** (decimals.externalDecimals - 18))`, then the function will return 0 and that would ultimately packed into the Deposit event. The Relayers are outside the scope of this audit but we assume they would not produce any currency on the other side of the bridge for a deposit of 0.

Impact Users can lose funds.

Recommendation The contract should revert when the converted value is 0 because the amount requested to bridge is too small.

Developer Response Applied the recommended fix.

4.1.23 V-CHAIN-VUL-023: getRoleMemberIndex() result is off by 1

Severity	Info	**Commit**	609ca8b
Type	Logic Error	**Status**	Fixed
File(s)	`contracts/utils/AccessControl.sol`		
Location(s)	getRoleMemberIndex()		
Confirmed Fix At	N/A		

The `AccessControl` contract provides the `getRoleMemberIndex()` function to return the index where the given account is stored for the given role.

```
function getRoleMemberIndex(bytes32 role, address account) public view returns (
    uint256) {
    return _roles[role].members._inner._indexes[bytes32(uint256(uint160(account)))];
}
```

Snippet 4.26: Definition of `AccessControl.getRoleMemberIndex()`

Here the reference `_roles[role].members._inner` refers to an instance of `struct Set` defined in `@openzeppelin/contracts/utils/structs/EnumerableSet.sol`. The documentation for `Set._indexes` indicates that it maps each value to the "Position of the value in the `values` array, plus 1 because index 0 means a value is not in the set." Hence the value returned by `getRoleMemberIndex()` is actually 1 greater than the index where the value is stored which implies `getRoleMember(R, getRoleMemberIndex(R,A)) != A`.

There are no uses of this function within the scope of this audit but the auditors believe the expected behavior would be `getRoleMember(R, getRoleMemberIndex(R,A)) == A`.

Impact Unexpected index may be returned which could end up allowing access in a case where it should not be allowed, depending on actual usage.

Recommendation `getRoleMemberIndex()` should subtract 1 from the value before returning.

Developer Response Developers determined the function is not needed and replaced the `AccessControl` contract entirely with the OpenZeppelin implemenation.

4.1.24 V-CHAIN-VUL-024: Handle empty input to poseidon_hash_fq_array

Severity	Info	**Commit**	ba3850e
Type	Maintainability	**Status**	Fixed
File(s)	`lightclient-circuits/src/poseidon.rs`		
Location(s)	multiple		
Confirmed Fix At	N/A		

In `lightclient-circuits/src/poseidon.rs` the `fq_array_poseidon()` and `poseidon_hash_fq_array()` functions expect a collection of items to perform the Poseidon hash over. In both functions, if the input contains no elements, the function will panic with *"called Option::unwrap() on a None value"* at the statement `current_poseidon_hash.unwrap()`.

```
let mut current_poseidon_hash = None;
for (i, chunk) in limbs.chunks(POSEIDON_SIZE - 2).enumerate() {
    poseidon.update(chunk);
    if i != 0 {
        poseidon.update(&[current_poseidon_hash.unwrap()]);
    }
    let _ = current_poseidon_hash.insert(poseidon.squeeze(ctx, gate));
}
Ok(current_poseidon_hash.unwrap())
```

Snippet 4.27: Excerpt from `fq_array_poseidon()`

Impact Panics on empty input with an unhelpful error message.

Recommendation Document why the empty input is not allowed and add an explicit assertion for it.

Developer Response Applied the recommended fix.

5 Fuzz Testing

5.1 Methodology

Our goal was to fuzz test Spectre, the Zero-Knowledge coprocessor within Sygma, to assess its functional correctness (i.e, whether the implementation deviates from the intended behavior) and potential crash states (i.e. whether it is possible to crash the process). We used AFL (american fuzzy lop) as our fuzzer and wrote tests to setup and call several important functions within Spectre. For functions implementing a common algorithm with a pre-existing Rust implementation, we used that implementation as an oracle to verify that the Spectre function produces the expected result. We used `https://github.com/axiom-crypto/pse-poseidon` for the Poseidon oracle and `https://crates.io/crates/sha2` for the Sha256 oracle.

5.2 Functions Fuzzed

Table 5.1 lists the functions we fuzz-tested in the first column. The second column shows the number of compute-minutes that the fuzzer spent testing the function and the third indicates the number of bugs identified while fuzzing the function.

Table 5.1: Fuzzing Summary.

Function Fuzzed	Minutes Fuzzed	Bugs Found
gadget::crypto::sha256_wide::Sha256ChipWide::digest	8,812	2
gadget::crypto::sha256_flex::Sha256Chip::digest	24,502	0
poseidon::fq_array_poseidon	20,820	1
ssz_merkle::ssz_merkleize_chunks	12,410	1
committee_update_circuit::CommitteeUpdateCircuit::synthesize	13,354	0
EndianConversions.toLittleEndian EndianConversions.toLittleEndian64 [a]	2,514	0
TOTAL	82,412	4

[a] These are from the `contracts` folder of the Spectre repository and are written in Solidity. The body of each function was copied over to a rust function and the `ethnum` crate was used to support the Solidity types.

The Veridise auditors devoted a total of 1,373 compute-hours to fuzzing this protocol, identifying a total of 4 bugs which are described in V-CHAIN-VUL-007, V-CHAIN-VUL-008, V-CHAIN-VUL-017, and V-CHAIN-VUL-024.