Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Succinct

SP1

Veridise Inc.
July 18, 2024

► **Prepared For:**

Succinct
https://www.succinct.xyz/

► **Prepared By:**

Shankara Pailoor
Tim Hoffman
Tyler Diamond

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

► **Version History:**

July 18, 2024	V2
July 08, 2024	V1
June 25, 2024	Initial Draft

Contents

1 Executive Summary

Succinct engaged Veridise to audit the security of two codebases: sp1-contracts and the recursion circuits in SP1. The review was broken up into three sub-audits which took place over the course of 7 weeks. The first audit focused on SP1's recursive verifier and lasted from June 03, 2024 to June 17, 2024. The goal of this audit was to review changes made to the overall logic after a previous informal audit Veridise performed a few months prior. Veridise conducted the assessment over 4 person-weeks, with 2 engineers reviewing code over 2 weeks on commit `1049583`, focusing on the differences between the original commits reviewed: namely commits `b6eac88`, `ab103ee`, and `02c6ea6`. The auditing strategy involved an extensive manual analysis of the source code performed by Veridise engineers. Since the changes made to the recursion circuits should have had corresponding changes made to the non-recursive verifiers and vice-versa, Veridise auditors checked to see if the two sets of verifiers were consistent. In addition, the auditors compared each circuit's logic (which verifies STARK proofs) to an existing implementation of a STARK verifier; specifically, the Plonky3 FRI implementation which each circuit was based on.

Next, on July 05, 2024, Veridise reviewed the Solidity smart contracts that can be deployed for verifying SP1 proofs. This review required 2 engineers to review the code over 1 day on commit 73c2a8d on the repository sp1-contracts.

Finally, on July 14, 2024, Veridise auditors reviewed additional changes made by Succinct to the recursion circuits. This review required 2 engineers to review code over 4 days on commit 6e968e3.

Project summary. (Recursion) At a high level, the SP1 recursion circuit takes as input SP1 STARK proofs and generates a groth-16 proof (the recursive proof) attesting to the fact that the STARK proofs were verified. Writing such recursion circuits is common practice among zkVMs to compress/compose multiple proofs so they can be verified efficiently. To write the circuit, SP1 engineers developed a circuit IR as well as a builder interface to write circuits in the IR. In addition, they wrote a compiler to compile the circuits into their own circuit assembly language which, in turn, can be translated into circuits in other frameworks like Gnark. Veridise engineers were asked to re-audit the SP1 recursion circuit, focusing on the changes made to the circuit since the original audit.

In particular, in commit `1049583`, the major changes made to the codebase were the following:

- In `circuit/` the main changes were in `circuit/stark.rs` and `circuit/poseidon2.rs`. In `stark.rs`, new checks were added to ensure the recursive verifier was in line with the core SP1 verifier. In `poseidon2.rs`, the implementation was refactored to operate over the BabyBear field whereas the old implementation was over BN254.
- In `program/`, the recursive verifier in `recursion/program/` was refactored into 4 distinct verifiers: a `core` verifier which verifies RISC-V execution traces, a `compress` verifier which verifies recursion proofs, a `deferred` verifier which verifies compressed SP1 proofs,and a `root` verifier which verifies the entrie tree of proofs. The main focus in this audit was

ensuring that each verifier was enforcing the proper checks on the inputs, the circuit was properly constrained, and public inputs were correctly committed to.

- Most of the code in `recursion/gnark-ffi/go/sp1/` was not in scope for the previous audit and was reviewed in the current one. The code was implementing BabyBear arithmetic in Gnark over BN254. The main concerns were making sure the computation didn't have any overflow issues i.e, the computation exceeds the bounds of the BabyBear modulus but is within the BN254 field.

Subsequently, in commit 6e968e3, the following major changes were made:

- Shards no longer need to have a CPU table as the prover now supports off-chip computation like GPU acceleration or custom precompiled circuits. As a result, the circuit now distinguishes between executable shards which are generated from CPU execution and non-executable shards which are not.
- The prover supports sharded memory to speed up proving time. This requires keeping track of address initialized and finalized for each shard.

Project summary. (sp1-contracts) The `sp1-contracts` repository consists of a set of Solidity contracts which are used to verify SP1 proofs. The main contract is `SP1VerifierGateway.sol` which maintains a set of SP1 verifiers. At a high level, SP1 verifiers can be registered through the gateway and users can supply proofs for a specific SP1 project and the gateway will call the appropriate verifier to verify the proofs.

Code assessment. (Recursion) The SP1 developers provided the SP1 recursion source code for review. Most of the code appears to be originally designed and written by SP1 developers; however, significant portions of the recursion circuits are based off Plonky3's implementation of FRI. The code base contains some documentation in the form of comments on functions and data structures. To facilitate the Veridise auditors' understanding of the code, the SP1 developers had a kick-off meeting with the Veridise auditors to walk through different sections of the code, particularly the changes made since the previous audit.

The source code contained a test suite, which the Veridise auditors noted covered many sections of the intended behavior of the verifier and prover. However, the auditors feel the test suite can be improved in two ways. First, there should be tests which cover the case where the verifier handles multiple proof shards. Such tests would have quickly caught V-SP1-VUL-001 as the verifier's logic was incorrect whenever multiple shards were verified. Second, the test suite could contain "negative" tests which try and check that a malicious prover cannot verify an invalid proof. Tests of this type are very important for ZK applications as they can be used to catch any major missing constraints.

The code was also well commented for the most part; however, in parts of the `gnark-ffi` code, implicit assumptions were not well documented such as in V-SP1-VUL-007 where the output of the Poseidon hash was not reduced to the BabyBear field; this was intentional but not documented. V-SP1-VUL-009 describes a case where the `MulFConst` instruction appeared to multiply a field element by an arbitrary constant, but in fact the code assumed that the constant was constrained to be less than 16. While such assumptions were correct, the lack of documentation could be a source of future bugs.

Finally, we observed the code base used a large number of magic constants (see V-SP1-VUL-015) even in cases where a corresponding global variable for that constant existed. The use of magic constants can be a source of bugs especially when the underlying value represented by the constant changes.

Code assessment. (sp1-contracts) The sp1-contracts also seemed to be originally designed and written by the SP1 developers and their implementations are very simple as they essentially hash the public inputs and then call the verifier contract to verify a zk proof. The code was well commented and specified the expected behavior along with any implicit assumptions made by the developers. The contracts also came with a test suite which exercised all external methods and tested both expected success and failure scenarios.

While the contracts themselves were well written, the code that generated the verifier contracts, namely `recursion/gnark-ffi/go/sp1/build.go` can be improved. For example, V-SP1-VUL-003 describes an issue where the type of verifier being generated depends on whether the string "dev" appears in a data directory as opposed to explicitly requiring the user to specify the working environment. As such, a test verifier with an insecurely generated structured reference string could be used in a production setting. Moreover, all the code in the file is written within a single function that is nearly 200 lines long. This should be refactored so that repeated functionality is captured in individual functions.

Summary of issues detected. The audit uncovered 19 issues, 5 of which are assessed to be of high or critical severity by the Veridise auditors. Specifically, V-SP1-VUL-001 describes an issue where the `core` verifier incorrectly handled multiple shard proofs. As such, any attempt to verify multiple shards would fail. V-SP1-VUL-002 describes a case where the initial sponge state of the challenger was left unconstrained. By allowing the attacker to control the initial state of the sponge, it becomes much easier for them to manipulate the output of the challenger. The Veridise auditors also identified 1 medium-severity issue, V-SP1-VUL-006, where the `core` verifier did not check the shard indices passed in. As such, a malicious prover could potentially prove more than the maximum number of shards permitted (2^{16}). The audit also uncovered 2 low-severity issues, 3 warnings, and 8 informational findings. The SP1 developers quickly acknowledged issues during weekly syncs, and are fixing all of the medium and high issues.

Recommendations. While the audited code was well written and had several end-to-end functional tests, the auditors observed several places where documentation and testing could be improved as described in the Code assessment. (Recursion) section above. Documentation and tests must be updated to accurately reflect the code as development continues on this codebase. Finally, since large parts of the recursion circuit were based on Plonky3's FRI implementation, the auditors recommend that the developers watch for any bug reports involving the Plonky3 FRI implementation as such bugs could also appear in the recursion circuit.

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
SP1	`1049583`	Rust	Desktop
sp1-contracts	73c2a8d	Solidity	EVM
SP1	6e968e3	Rust	Desktop

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
June 03 - June 17, 2024	Manual	2	4 person-weeks
July 05 - July 06, 2024	Manual	2	2 person-days
July 14 - July 18, 2024	Manual	2	4 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	5	5	4
Medium-Severity Issues	1	1	1
Low-Severity Issues	2	2	0
Warning-Severity Issues	3	3	1
Informational-Severity Issues	8	8	3
TOTAL	19	19	9

Table 2.4: Category Breakdown.

Name	Number
Logic Error	6
Maintainability	5
Data Validation	3
Access Control	3
Documentation	1
Gas Optimization	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was initially scoped to provide a security assessment of SP1's recursion-related source code, particularly the security of the code changes made after the first audit. Afterwards, the audit was exapnded to review the corresponding smart contracts. Overall in this audit we sought to answer the following questions:

- Do the recursive verifiers correctly commit to the public values used in the proofs?
- Are the inputs to the proofs properly validated?
- Do the recursion circuits contain any underconstrained vulnerabilities?
- Does the `gnark-ffi` code properly perform BabyBear arithmetic within the BN254 prime? In particular, are bounds being properly checked?
- Are the recursion circuits' logic consistent with existing STARK verifiers?
- Are the cryptographic primitives used in the circuits implemented correctly?
- Are the SP1 verifiers generated correctly? (sp1-contracts)
- Are there any centralization risks associated with the contracts (sp1-contracts)

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved an extensive manual analysis by Veridise auditors.

Scope. The scope of the re-audit of the `recursion/` folder provided by the SP1 developers was limited to the `compiler, program, and circuit` sub folders of `recursion/`. The purpose of each subfolder is as follows:

- **Compiler**: Details the intermediate language in which the recursive prover is written, and exposes methods for building out a recursively-verified program.
- **Program**: Implements a recursion circuit in the circuit DSL which verifies an SP1 STARK proof.
- **Circuit**: Implements a recursion circuit that is optimized for eventual compilation into Gnark.

The audit of the `sp1-contracts/` repository was limited to the `src/contracts/` sub-directory which contained the verifier interface along with their verifier gateway contract which is used to manage and discharge proofs to different sp1 verifiers. The code to generate the verifiers from the circuit was located in the sp1 repository under `recursion/gnark-ffi/assets/build.go` `recursion/`

Methodology. Veridise auditors first reviewed their prior audit of the code base and inspected the source code diff between the previous version and the one in this audit. During the audit, Veridise auditors regularly met with the SP1 developers to ask questions about the code and would frequently message the developers over Slack.

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-SP1-VUL-001	verify_shard takes incorrect indexes	High	Fixed
V-SP1-VUL-002	Sponge state not set to 0	High	Fixed
V-SP1-VUL-003	Wrong SRS can be chosen	High	Acknowledged
V-SP1-VUL-004	Incorrect checks on deferred digest	High	Fixed
V-SP1-VUL-005	Incorrect zero check of deferred_proof_digest	High	Fixed
V-SP1-VUL-006	verify_shard does not range check shard indices	Medium	Fixed
V-SP1-VUL-007	Gnark Poseidon Hash not reduced	Low	Acknowledged
V-SP1-VUL-008	Centralization Risk	Low	Open
V-SP1-VUL-009	Unbalanced handling of I/O streams	Warning	Fixed
V-SP1-VUL-010	MulFConst documented incorrectly	Warning	Acknowledged
V-SP1-VUL-011	Two-step ownership is not used	Warning	Acknowledged
V-SP1-VUL-012	Comments with typos and outdated information	Info	Fixed
V-SP1-VUL-013	Excess visibility	Info	Fixed
V-SP1-VUL-014	Dead code	Info	Fixed
V-SP1-VUL-015	Use of magic constants	Info	Acknowledged
V-SP1-VUL-016	Define() may panic	Info	Acknowledged
V-SP1-VUL-017	Performance	Info	Acknowledged
V-SP1-VUL-018	Duplicate name of variable and package	Info	Acknowledged
V-SP1-VUL-019	Directory creation called after usage	Info	Acknowledged

4.1 Detailed Description of Issues

4.1.1 V-SP1-VUL-001: verify_shard takes incorrect indexes

Severity	High	**Commit**	1049583
Type	Logic Error	**Status**	Fixed
File(s)	`program/src/machine/{core`, compress, deferred, `root}.rs`		
Location(s)	multiple		
Confirmed Fix At	8c0f501		

The SP1 prover breaks up the proof of an execution trace into a collection of shard proofs which are proofs about one chunk of the execution. The shard proofs are then verified in aggregate by the verifier whereby each shard is itself verified via a call to `verify_shard` and the linking between each shard is also verified.

The `verify_shard` implementation takes as input a `shard_idx` variable which indicates which shard in the collection is being verified. This variable is effectively used as a boolean because it is expected that certain instructions like `MemoryFinalize` and `MemoryInit` are used in the shard iff `shard_idx = 1`. This is seen in the below code snippet:

The recursive prover, which implements the verification of the shards as a circuit, incorrectly assumes that the first shard in the batch has an index of 1. in general, the shards do not have to start at 1, but can be any value in $[0, 2^{16})$. As such, if the collection of shards passed in does not start at 1, then the set of constraints will likely become unsatisfiable as the `verify_shard` will assert that `MemoryFinalize` and `MemoryInit` be used even though they are not.

Impact The recursive verifier will not be able to produce a proof that a batch was verified for any batch of shards that does not start at index 1.

Recommendation Replace `shard_idx` in the caller with the shard's index.

Developer Response The developers fixed the issue by removing the parameter and extracting the shard index from the public values for the shard.

```
if chip.name() == "MemoryProgram" {
    builder.if_eq(shard_idx, C::N::one()).then_or_else(
        |builder| {
            builder.assert_var_ne(index, C::N::from_canonical_usize(EMPTY));
        },
        |builder| {
            builder.assert_var_eq(index, C::N::from_canonical_usize(EMPTY));
        },
    );
}

if chip.name() == "MemoryInit" {
    builder.if_eq(shard_idx, C::N::one()).then_or_else(
        |builder| {
            builder.assert_var_ne(index, C::N::from_canonical_usize(EMPTY));
        },
        |builder| {
            builder.assert_var_eq(index, C::N::from_canonical_usize(EMPTY));
        },
    );
}

if chip.name() == "MemoryFinalize" {
    builder.if_eq(shard_idx, C::N::one()).then_or_else(
        |builder| {
            builder.assert_var_ne(index, C::N::from_canonical_usize(EMPTY));
        },
        |builder| {
            builder.assert_var_eq(index, C::N::from_canonical_usize(EMPTY));
        },
    );
}
```

Snippet 4.1: Snippet from `verify_shard`

4.1.2 V-SP1-VUL-002: Sponge state not set to 0

Severity	High	**Commit**	1049583
Type	Logic Error	**Status**	Fixed
File(s)	`program/src/challenger.rs`		
Location(s)	new()		
Confirmed Fix At	N/A		

The `DuplexChallengerVariable`'s constructor, shown below, is expected to produce a challenger in a default state where the `sponge_state` is an array of zeros and `nb_inputs` and `nb_outputs` are zero.

```
pub fn new(builder: &mut Builder<C>) -> Self {
    DuplexChallengerVariable::<C> {
        sponge_state: builder.dyn_array(PERMUTATION_WIDTH),
        nb_inputs: builder.eval(C::N::zero()),
        input_buffer: builder.dyn_array(PERMUTATION_WIDTH),
        nb_outputs: builder.eval(C::N::zero()),
        output_buffer: builder.dyn_array(PERMUTATION_WIDTH),
    }
}
```

Snippet 4.2: Snippet from `new`

While the `nb_inputs` and `nb_outputs` are correctly set to zero, `sponge_state` is assigned an array of felts whose values can be chosen arbitrarily by the prover. Allowing an attacker to have complete control over the value set in the `sponge_state` gives them much more flexibility to manipulate the outputs during `duplexing`.

Recommendation Add constraints that assert that each element of `sponge_state` is 0 .

Developer Response Developers have fixed the issue in the following PR `https://github.com/succinctlabs/sp1/pull/951`.

4.1.3 V-SP1-VUL-003: Wrong SRS can be chosen

Severity	High	**Commit**	9494407
Type	Data Validation	**Status**	Acknowledged
File(s)	`gnark-ffi/go/sp1/build.go`		
Location(s)	Build()		
Confirmed Fix At	N/A		

The `Build()` function will construct the SRS depending upon whether the `dataDir` variable contains the `dev` string. If `dataDir` *does* contain the string, then an unsafe SRS is chosen that does not provide security to the proving system

```go
if !strings.Contains(dataDir, "dev") {
  ...
} else {
    srs, srsLagrange, err = unsafekzg.NewSRS(scs)
    ...
```

Snippet 4.3: Snippet from `build.go`

This means that if *any* directory along the filepath contains the string `dev` the unsafe SRS will be used as opposed to using the real one.

Impact The incorrect (unsafe) SRS can be chosen for proof deployment, and this will lead to breaking the soundness of verifiers that are deployed with this script.

Recommendation Pass a function parameter to `Build()`, or an environmental variable, to determine if the test SRS should be used.

Developer Response The developers have acknowledged the issue but do not plan on fixing it.

4.1.4 V-SP1-VUL-004: Incorrect checks on deferred digest

Severity	High	**Commit**	6e968e3
Type	Logic Error	**Status**	Fixed
File(s)	`program/src/machine/{compress.rs, core.rs}`		
Location(s)	verify		
Confirmed Fix At	N/A		

The recursive verifiers need to check that if there is a deferred digest (i.e not zero), then it remains the same across shards and is zero otherwise. However, the current implementation in the `compress` and `core` verifiers only check that the first word of the digest remains the same as seen below:

```
// If ‘deferred_proofs_digest‘ is not zero, then ‘public_values.deferred_proofs_digest
// should be the current value.
let is_zero: Var<_> = builder.eval(C::N::zero());
#[allow(clippy::needless_range_loop)]
for i in 0..deferred_proofs_digest.len() {
    let d = felt2var(builder, deferred_proofs_digest[i]);
    builder.if_ne(d, C::N::zero()).then(|builder| {
        builder.assign(is_zero, C::N::zero());
    });
}
builder.if_eq(is_zero, C::N::zero()).then(|builder| {
    builder.assert_felt_eq(
        deferred_proofs_digest[0],
        current_public_values.deferred_proofs_digest[0],
    );
});
```

Snippet 4.4: Snippet from `verify` in `compress`

Impact By not checking the other elements, the other elements of the digest can be varied while still satisfying the constraints. The concern is that an attacker could set the other digest values in a way that would allow the final digest to be equal to the end_reconstruct_digest during the completion check.

Recommendation Check that all the elements are the same.

Developer Response The developers have been acknowledged the issue.

4.1.5 V-SP1-VUL-005: Incorrect zero check of deferred_proof_digest

Severity	High	**Commit**	6e968e3
Type	Logic Error	**Status**	Fixed
File(s)	`program/src/machine/compress.rs`		
Location(s)	verify		
Confirmed Fix At	f11e51a		

The `compress` verifier performs a check (lines 473-479) to see if the previous `deferred_proofs_digest` variable is zero as shown below. If it isn't, then it asserts that the `deferred_proofs_digest` for the current shard must equal the previous one.

```
// If 'deferred_proofs_digest' is not zero, then 'public_values.
    deferred_proofs_digest
// should be the current value.
let is_zero: Var<_> = builder.eval(C::N::zero());
#[allow(clippy::needless_range_loop)]
for i in 0..deferred_proofs_digest.len() {
    let d = felt2var(builder, deferred_proofs_digest[i]);
    builder.if_ne(d, C::N::zero()).then(|builder| {
        builder.assign(is_zero, C::N::zero());
    });
}
builder.if_eq(is_zero, C::N::zero()).then(|builder| {
    builder.assert_felt_eq(
        deferred_proofs_digest[0],
        current_public_values.deferred_proofs_digest[0],
    );
});
```

Snippet 4.5: Snippet from `verify`

However, the zero check is not correct as it initializes the `is_zero` variable to be 0 instead of 1. Thus, the check always sets `is_zero` to be 0 even if the digest is actually zero.

Impact This forces all `deferred_proofs_digest` to be the same even if they aren't.

Recommendation `is_zero` should be initialized to 1 not 0.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.6 V-SP1-VUL-006: verify_shard does not range check shard indices

Severity	Medium	**Commit**	1049583
Type	Data Validation	**Status**	Fixed
File(s)	`program/src/machine/core.rs`		
Location(s)	verify()		
Confirmed Fix At	N/A		

The SP1 Verifier is designed to only verify at most 2^{16} shards; as such, each shard index should lie in the half-open interval $[0, 2^{16})$. However, the `verify()` procedure in `core.rs` does not perform this range check on the shard indices.

Impact The recursive verifier allows proofs to be verified that are larger than the max number of shards.

Recommendation Add a range check on each shard index to ensure it is within $[0, 2^{16})$.

Developer Response Developers have acknowledged the issue.

4.1.7 V-SP1-VUL-007: Gnark Poseidon Hash not reduced

Severity	Low	**Commit**	1049583
Type	Data Validation	**Status**	Acknowledged
File(s)	`gnark-ffi/sp1/poseidon2_babybear.go`		
Location(s)	PermuteMut()		
Confirmed Fix At	N/A		

The SP1 developers wrote an implementation of the Poseidon hash function which in turn uses field primitives such as `addF` and `mulF` implemented in `babybear.go` which (respectively) add and multiply field elements. To optimize performance, the developers have implemented each arithmetic operation to allow the resulting value to exceed the field modulus and only reduce the value at a certain threshold. As such, the developers should make sure to note whether a function's result is intended to be non-reduced so that users of the function are aware and can reduce the resulting value when necessary.

In particular, the hash implementation doesn't reduce the resulting sponge state and there is no documentation indicating that this was the intention. As such, a user of the Poseidon hash may forget to reduce the resulting state.

Impact See above.

Recommendation Add documentation indicating that the resulting state is not reduced.

Developer Response Developers stated that not reducing the state was intentional but will add documentation.

4.1.8 V-SP1-VUL-008: Centralization Risk

Severity	Low	**Commit**	73c2a8d
Type	Access Control	**Status**	Open
File(s)	`SP1VerifierGateway.sol`		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Similar to many projects, Succinct's `SP1VerifierGateway` declare an administrator role that is given special permissions. In particular, these administrators are given the following abilities:

- The owner can add malicious verifiers to routes
- The owner can freeze routes, preventing verification of proofs

Impact If a private key were stolen, a hacker would have access to sensitive functionality that could compromise the project. For example, a malicious route could enable dependent applications to verify invalid proofs that provide the hacker with unintended behavior.

Recommendation As these are all particularly sensitive operations, we would encourage the developers to utilize a decentralized governance or multi-sig contract as opposed to a single account, which introduces a single point of failure.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.9 V-SP1-VUL-009: Unbalanced handling of I/O streams

Severity	Warning	**Commit**	1049583
Type	Logic Error	**Status**	Fixed
File(s)	multiple		
Location(s)	multiple		
Confirmed Fix At	N/A		

At several locations, files are opened without being closed or closed more than once.

1. In `gnark-ffi/go/sp1/build.go`
   a) `solidityVerifierFile` is not closed
   b) `srsFile.Close()` is executed twice, once via the `defer` statement and once directly. This occurs for two distinct variables with this same name, created at lines 67 and 85.
   c) The file `srsLagrangeFileName` is opened at line 97 but is already available via the variable `srsLagrangeFile` defined at line 55. The version opened at line 97 is also closed twice.
2. In `gnark-ffi/go/sp1/prove.go`
   a) `scsFile` is not closed
   b) `pkFile` is not closed
   c) `vkFile` is not closed

Impact Writing to a file stream without subsequently closing it can lose data that has been buffered but not yet written when the program terminates.

Recommendation Ensure that each stream is closed exactly one time.

Developer Response Developers have applied the recommendation.

4.1.10 V-SP1-VUL-010: MulFConst documented incorrectly

Severity	Warning	**Commit**	1049583
Type	Documentation	**Status**	Acknowledged
File(s)	`gnark-ffi/go/sp1/babybear/babybear.go`		
Location(s)	MulFConst()		
Confirmed Fix At	N/A		

The `babybear.go` file exposes a function called `MulFConst`, shown below, that multiplies a variable with an integer. However, this function implicitly assumes that `0 <= b < 16` as the resulting felt is assigned `NbBits` equal to `a.NbBits + 4`. The developers informed the auditors that this assumption was, in fact, intended and callers should only pass in values for `b` which fit in the assumed range. Furthermore, the callers of this function seem to satisfy this assumption as all uses of this function pass in values within this range.

Nevertheless, not making this assumption explicit in comments and not checking whether `b` actually fits in this range can lead to issues down the road if new functionality is added which forgets this assumption.

```
func (c *Chip) MulFConst(a Variable, b int) Variable {
    return c.ReduceFast(Variable{
        Value:  c.api.Mul(a.Value, b),
        NbBits: a.NbBits + 4,
    })
}
```

Snippet 4.6: Definition of `MulFConst()`

Impact Callers can mistakenly call this function and pass in a value for `b` which is larger than `16`.

Recommendation Recommend changing the name of the function to `MulFSmallConst` and adding an explicit check that `b < 16`.

Developer Response Developers have acknowledged the issue.

4.1.11 V-SP1-VUL-011: Two-step ownership is not used

Severity	Warning	**Commit**	73c2a8d
Type	Access Control	**Status**	Acknowledged
File(s)	`SP1VerifierGateway.sol`		
Location(s)	See issue details		
Confirmed Fix At	N/A		

The `SP1VerifierGateway` contract inherits from OpenZeppelin's `Ownable` contract. This contract does require the receiving owner to acknowledge transfer of ownership.

Impact Specifying the wrong address for an ownership transfer will lead to loss of administrative functionality.

Recommendation Utilize ownership that requires the receiving owner to acknowledge the transfer of ownership, such as OpenZeppelin's `Ownable2Step` as seen here: `https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol`.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.12 V-SP1-VUL-012: Comments with typos and outdated information

Severity	Info	**Commit**	
Type	Maintainability	**Status**	Fixed
File(s)	multiple		
Location(s)	multiple		
Confirmed Fix At	N/A		

The code documentation has typos or appears to be outdated in several locations.

1. In `program/src/machine/core.rs`
   a) The documentation comment for `verify()` has several typos:
      i. reprersents → represents
      ii. shardf → shard
      iii. reconstructiing → reconstructing
      iv. chanllenger → challenger
      v. Therefoee → Therefore
      vi. assertds → asserts
      vii. deffered → deferred
   b) The `verify()` function contains a check if `shard == 1` but the comment above says "If the shard is zero" so that should be updated to match the implementation.

```
// If the shard is zero, verify the global initial conditions hold on challenger and
    pc.
let shard = felt2var(builder, public_values.shard);
builder.if_eq(shard, C::N::one()).then(|builder| {
```

Snippet 4.7: Excerpt from `verify()`

   c) On line 404 of `verify()` within `compress.rs` the following the comment `// Assert that the leaf challenger is always the same.` doesn't have corresponding code underneath which implements it. Instead, the relevant code is on line 422 and so the comment or code should be moved so they remain consistent.

Impact Unclear and/or outdated documentation makes the project harder to maintain.

Recommendation Apply the changes mentioned above.

Developer Response Developers have applied the recommendation.

4.1.13 V-SP1-VUL-013: Excess visibility

Severity	Info	Commit	1049583
Type	Access Control	Status	Fixed
File(s)	multiple		
Location(s)	multiple		
Confirmed Fix At	N/A		

The visibility modifiers should be more strict in the following locations:

1. `gnark-ffi/go/sp1/babybear/babybear.go`
   a) Global value `MODULUS` should be private
   b) Function `NegF()` should be private
   c) Function `InvF()` should be private
   d) Function `ReduceFast()` should be private
   e) Function `ReduceWithMaxBits()` should be private
   f) Function `ReduceHint()` should be private
   g) Function `InvFHint()` should be private
   h) Function `invEHint()` should be private
2. `gnark-ffi/go/sp1/poseidon2/constants.go`
   a) Global value `RC3` should be private
   b) Global value `RC16` should be private
3. `gnark-ffi/go/sp1/poseidon2/poseidon2_babybear.go`
   a) Global value `BABYBEAR_NUM_EXTERNAL_ROUNDS` should be private
   b) Global value `BABYBEAR_NUM_INTERNAL_ROUNDS` should be private
4. `gnark-ffi/go/sp1/poseidon2/poseidon2.go`
   a) Global value `WIDTH` should be private
   b) Global value `NUM_EXTERNAL_ROUNDS` should be private
   c) Global value `NUM_INTERNAL_ROUNDS` should be private
   d) Global value `DEGREE` should be private
   e) Function `AddRc()` should be private
   f) Function `SboxP()` should be private
   g) Function `Sbox()` should be private
5. `gnark-ffi/go/sp1/poseidon2/utils.go`
   a) Function `DiffusionPermuteMut()` should be private
   b) Function `MatrixPermuteMut()` should be private
6. `gnark-ffi/go/sp1/sp1.go`
   a) Global value `SRS_FILE` should be private
   b) Global value `SRS_LAGRANGE_FILE` should be private
   c) Global value `CONSTRAINTS_JSON_FILE` should be private
   d) Global value `VERIFIER_CONTRACT_PATH` should be private
   e) Global value `CIRCUIT_PATH` should be private
   f) Global value `VK_PATH` should be private
   g) Global value `PK_PATH` should be private

7. `program/src/machine/utils.rs`
   a) Function `calculate_public_values_digest()` should be private

Impact Excess visibility can create security risks such as modifying the internal state of a structure in an inconsistent manner.

Recommendation Always apply the principle of least privilege when setting visibility modifiers. The violations mentioned above consider only uses within the `sp1` repository. Before making the visibility adjustments, the developers must also consider external uses that were not disclosed to the auditors.

Developer Response Developers have applied the recommendation.

4.1.14 V-SP1-VUL-014: Dead code

Severity	Info	**Commit**	1049583
Type	Maintainability	**Status**	Fixed
File(s)	multiple		
Location(s)	multiple		
Confirmed Fix At	N/A		

The following definitions are not used after being defined:

1. `gnark-ffi/go/sp1/babybear/babybear.go`
   a) `var W`
2. `gnark-ffi/go/sp1/poseidon2/poseidon2.go`
   a) `Poseidon2Chip.one`
3. `gnark-ffi/go/sp1/poseidon2/poseidon2_babybear.go`
   a) `const BABYBEAR_DEGREE`
4. `gnark-ffi/go/sp1/sp1.go`
   a) `var WITNESS_JSON_FILE` (the developers may have intended to use this in `build.go`)

Impact Definitions that are not used make the code messy and harder to maintain.

Recommendation Remove the unused definitions.

Developer Response Developers have applied the recommendation.

4.1.15 V-SP1-VUL-015: Use of magic constants

Severity	Info	**Commit**	1049583
Type	Maintainability	**Status**	Acknowledged
File(s)	multiple		
Location(s)	multiple		
Confirmed Fix At	N/A		

The following locations use constant values whose meaning is not made clear:

1. `gnark-ffi/go/sp1/babybear/babybear.go`
   a) In `ReduceFast()`, the value `120`
   b) In `MulE()`, the value `11`
2. `gnark-ffi/go/sp1/sp1.go`
   a) The `"Permute"` case uses the constant `3` but should use `poseidon2.WIDTH` instead. Additionally, the implementation here should use two loops like the `"PermuteBabyBear"` case does to be future-proof against changes to `poseidon2.WIDTH`.
   b) The `"PermuteBabyBear"` case uses the value `16` in three locations but should use `poseidon2.BABYBEAR_WIDTH` instead.
3. `program/src/machine/deferred.rs`
   a) In the snippet below, the value `16` should be `2 * DIGEST_SIZE`

```
builder.set(&mut poseidon_inputs, j * WORD_SIZE + k + 16, element);
```

Snippet 4.8: Snippet from `verify()`

4. `program/src/hints.rs`
   a) The last two parameters of `DuplexChallenger` should be `PERMUTATION_WIDTH, HASH_RATE` instead of `16, 8`. This occurs on the line `impl Hintable<C> for DuplexChallenger<InnerVal, InnerPerm, 16, 8>` and three occurrences of the expression `DuplexChallenger::<InnerVal, InnerPerm, 16, 8>::read(builder)`.
5. `program/src/stark.rs`
   a) In `verify_shard()`, the loop over the machine chips uses the string literals `"CPU"`, `"MemoryProgram"`, `"MemoryInit"`, and `"MemoryFinalize"` for the cases. These same string literals are used in the Chip definitions and elsewhere in the code. New constants should be declared for these string literals and all uses replaced with the constants.
6. `program/src/machine/core.rs`
   a) Every shard index should be between $[0, 2^{16})$ and this is enforced via the range check The range check on each shard is as follows: `builder.range_check_f(public_values.shard, 16);` which is correct but the constant `16` should be replaced with a constant like `LOG_MAX_SHARD_IDX`.

Impact The meaning of an unnamed constant value is unclear.

Recommendation Add a comment documenting the meaning of the constant value or extract the value into a constant declaration with a descriptive name.

Developer Response Developers have acknowledged the issues and plan to fix in the future.

4.1.16 V-SP1-VUL-016: Define function could panic instead of returning an error

Severity	Info	**Commit**	1049583
Type	Logic Error	**Status**	Acknowledged
File(s)	`gnark-ffi/go/sp1/sp1.go`		
Location(s)	Define()		
Confirmed Fix At	N/A		

The `"Num2BitsV"` and `"Num2BitsF"` cases of the `Define()` function convert an input value to a bit array and then assign the elements of that bit array to positions in the `vars` map as defined by the values of `cs.Args[0][i]`.

```
case "Num2BitsV":
    numBits, err := strconv.Atoi(cs.Args[2][0])
    if err != nil {
        return fmt.Errorf("error converting number of bits to int: %v", err)
    }
    bits := api.ToBinary(vars[cs.Args[1][0]], numBits)
    for i := 0; i < len(cs.Args[0]); i++ {
        vars[cs.Args[0][i]] = bits[i]
    }
case "Num2BitsF":
    bits := fieldAPI.ToBinary(felts[cs.Args[1][0]])
    for i := 0; i < len(cs.Args[0]); i++ {
        vars[cs.Args[0][i]] = bits[i]
    }
```

Snippet 4.9: Snippet from `Define()`

Both loops that perform the copies have iteration count equal to `len(cs.Args[0])` but do not check if the `bits` array contains that many elements. If `len(cs.Args[0]) > len(bits)`, the program will panic with "runtime error: index out of range" rather than returning an `error` from the `Define()` function. In the `"Num2BitsV"` case, `len(bits)==numBits`. In the `"Num2BitsF"` case, `len(bits)==32`.

Impact The program will terminate abruptly rather than returning an `error` which can be handled.

Recommendation Add checks for these conditions that return an `error`.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.17 V-SP1-VUL-017: Performance

Severity	Info	**Commit**	1049583
Type	Gas Optimization	**Status**	Acknowledged
File(s)	multiple		
Location(s)	multiple		
Confirmed Fix At	N/A		

Performance could be improved in the following locations:

1. `gnark-ffi/go/sp1/babybear/babybear.go`
   a) In the snippet below from `ReduceWithMaxBits()`, the final line performs an array access when the value is already stored in the local variable `remainder`. Avoid the array access by using `remainder` in the final line shown in the snippet.

```
    remainder := result[1]
    p.rangeChecker.Check(remainder, 31)
    p.api.AssertIsEqual(x, p.api.Add(p.api.Mul(quotient, MODULUS), result[1]))
```

Snippet 4.10: Snippet from `ReduceWithMaxBits()`

2. `gnark-ffi/go/sp1/poseidon2/utils.go`
   a) In the snippet below from `MatrixPermuteMut()`, two calls to `Add()` are used to compute the sum of three values. However, `Add()` is variadic so all three values could be passed into a single call to avoid unnecessary intermediate structures.
3. `program/src/machine/core.rs`
   a) The `verify()` function keeps a reference to the exit code from the first batch and then generates assertions that every other batch has that same exit code. However, for each batch it also adds an assertion that the exit code from the first batch is zero.

```
// Assert that exit code is the same for all proofs.
builder.assert_felt_eq(exit_code, public_values.exit_code);

// Assert that the exit code is zero (success) for all proofs.
builder.assert_felt_eq(exit_code, C::F::zero());
```

Snippet 4.11: Snippet from `verify()`

   One assert for each batch could be avoided by directly asserting that every batch exit code equals zero.
4. `program/src/utils.rs`
   a) In `get_preprocessed_data()`, the `clone()` of the chip name is not necessary.
5. `program/src/fri/mod.rs`
   a) In the snippet below, the second call to `clone()` is not necessary.
6. `program/src/stark.rs`

```
builder.poseidon2_compress_x(
    &mut root.clone(),
    &root.clone(),
    &next_height_openings_digest,
);
```

Snippet 4.12: Snippet from `verify_batch()`

a) In `verify_shard()`, the loop over the machine chips contains multiple `if` statements to check the value of the chip name. The conditions checked are mutually exclusive of one another so they should use else branches to avoid checking the remaining conditions once a match is found. In fact, all of the `"Memory*"` conditions have identical bodies so the snippet below could be used.

```
if chip.name() == "CPU" {
    builder.assert_var_ne(index, C::N::from_canonical_usize(EMPTY));
} else if chip.name() == "MemoryProgram"
    || chip.name() == "MemoryInit"
    || chip.name() == "MemoryFinalize"
{
    builder.if_eq(shard_idx, C::N::one()).then_or_else(
        |builder| {
            builder.assert_var_ne(index, C::N::from_canonical_usize(EMPTY));
        },
        |builder| {
            builder.assert_var_eq(index, C::N::from_canonical_usize(EMPTY));
        },
    );
}
```

Snippet 4.13: Suggested code for the chip name conditions

7. `circuit/src/fri.rs`
   a) The clone is not necessary in the following lines, in the second a reference is needed when the clone is removed:
      i. `for (batch_opening, round) in izip!(query_opening.clone(), &rounds)`
      ii. `for (mat_opening, mat) in izip!(batch_opening.opened_values.clone(), mats )`
      iii. `for (p_at_x, &p_at_z) in izip!(mat_opening.clone(), ps_at_z)`
      iv. `let index_sibling: Var<_> = builder.eval(one - index_bits.clone()[offset ])`

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.18 V-SP1-VUL-018: Duplicate name of variable and package

Severity	Info	**Commit**	9494407
Type	Maintainability	**Status**	Acknowledged
File(s)	`gnark-ffi/go/sp1/build.go`		
Location(s)	Build()		
Confirmed Fix At	N/A		

The `build.go` file imports `github.com/consensys/gnark/frontend/cs/scs`, which brings the module in scope and can therefore be referenced as `scs`. However, a variable is declared with the same name:

```
scs, err := frontend.Compile(ecc.BN254.ScalarField(), scs.NewBuilder, &circuit)
```

Snippet 4.14: Snippet from `build.go`

Impact Although the correct code execution is performed, this introduces unnecessary confusion and can harm maintainability for the future.

Recommendation Use a different name for the locally defined `scs` variable.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.19 V-SP1-VUL-019: Directory creation called after usage

Severity	Info	**Commit**	9494407
Type	Maintainability	**Status**	Acknowledged
File(s)	`gnark-ffi/go/sp1/build.go`		
Location(s)	Build()		
Confirmed Fix At	N/A		

The `Build()` function requires reading or writing to various files and subdirectories in the `dataDir` variable. One example is the `srsLagrangeFileName`:

```
srsLagrangeFileName := dataDir + "/" + srsLagrangeFile
srsLagrangeFile, err := os.Create(srsLagrangeFileName)
if err != nil {
        log.Fatal("error creating srs file: ", err)
        panic(err)
}
```

Snippet 4.15: Snippet from `build.go`

Later in the `build.go` file, the call to actually make this directory is defined:

```
os.MkdirAll(dataDir, 0755)
```

Snippet 4.16: Snippet later in `build.go`

Impact Delaying the creation of the directory may cause the `build.go` script to panic prematurely.

Recommendation Move the directory creation call before any of the file reading and writing is performed.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.