Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Succinct

gnark-plonky2-verifier

and

SuccinctGateway

Veridise Inc.

February 21, 2024

► **Prepared For:**

Succinct
https://succinct.xyz

► **Prepared By:**

Benjamin Sepanski
Ian Neal
Sorawee Porncharoenwase
Tim Hoffman
Kostas Ferles

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

► **Version History:**

Feb. 20, 2024	V3
Jan. 11, 2024	V2
Jan. 10, 2024	V1
Nov. 30, 2023	Initial Draft

Contents

1 Executive Summary

From Nov. 6, 2023 to Nov. 27, 2023, Succinct engaged Veridise to review the security of their gnark-plonky2-verifier project. The review covered the circuits implementing a plonky2 verifier. The circuits were implemented using the gnark zk-SNARK library*, which is written in the Go language. Veridise conducted the assessment over 12 person-weeks, with 4 engineers reviewing code over 3 weeks on commit `0x89b5a01e`.

Closely following the circuit review, from Jan. 4 to Jan. 5, 2024, Succinct further engaged Veridise to review the security of their SuccinctGateway. This review covered two smart contracts written in Solidity inside of the succinctx repository†. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Code assessment. The gnark-plonky2-verifier developers provided the source code of the gnark-plonky2-verifier circuits for review. To facilitate the Veridise auditors' understanding of the code, the gnark-plonky2-verifier developers provided a high-level overview of the implemented circuits as well as several references to material from the original plonky2 verifier‡. Following the audit, 2 minor changes to gnark-plonky2-verifier were also reviewed by Veridise, adding a small feature and slight optimizations.

Generally, the gnark-plonky2-verifier closely mirrored the original implementation of the plonky2 verifier so our auditing team could easily spot any deviations from the reference implementation. Except for some missing features, the Veridise auditors spotted very few omissions/deviations from the reference implementation. The source code also contained some documentation in the form of READMEs and documentation comments on core functions and structs.

The SuccinctGateway developers also provided the source code. These contracts are original to Succinct. The SuccinctGateway allows users to designate certain smart contracts as "verifier contracts," then request whitelisted sets of provers to perform on-chain calls with inputs which satisfy the verifiers. The developers worked with Veridise to provide additional information about the behavior and intended usage, answering several questions about the protocol.

The source code contained a test suite. The tests across the entire suite covered, on average, 79.1% of the gnark-plonky2-verifier program statements under test (with a standard deviation of 22.5), and 71.2% of the contract statements.

Summary of issues detected. The audit uncovered 39 issues, 3 of which are assessed to be of high or critical severity by the Veridise auditors. All three of these issues (V-SCT-VUL-001,

* `https://github.com/succinctlabs/gnark-plonky2-verifier/tree/89b5a01e4b`
† `https://github.com/succinctlabs/succinctx/tree/abd43565`
‡ `https://github.com/0xPolygonZero/plonky2/tree/64cc1000e7`

V-SCT-VUL-002, and V-SCT-VUL-003) were caused by missing constraints on values provided by hints[§]. No issues of high or critical severity were identified inside the smart contracts.

The Veridise auditors also identified several medium-severity issues, including a missing range check in the evaluation proofs (V-SCT-VUL-005), the improper use of a function which may mutate its arguments (V-SCT-VUL-004), and the possibility of locked funds in the contract (V-SCT-VUL-005). Finally, the audit uncovered a number of minor issues, including slightly over-constrained circuits (V-SCT-VUL-007, V-SCT-VUL-012), ignored parameters (V-SCT-VUL-033), minor logical errors (V-SCT-VUL-009, V-SCT-VUL-010, V-SCT-VUL-028) and various maintainability issues.

As of commits `0xc01f530fe` (gnark-plonky2-verifier) and `0x553d44b6` (SuccinctGateway), Succinct has fixed all of the identified issues in the gnark-plonky2-verifier. The SuccinctGateway developers have also provided fixes to all of the identified issues. Only the warning issue V-SCT-VUL-026 remains partially unresolved. However, any centralization risk to the users has been mitigated by allowing custom prover whitelists. The only remaining risk is left to the contract operators, as the fee vault may be changed if the owner is compromised, or users who choose to rely on the default prover list.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve the gnark-plonky2-verifier.

Extended type system for field emulation. Succinct's type system does much to improve the readability and safety of their circuits. One key distinction which is not included in the type system, but could greatly simplify reasoning about the code, is distinguishing between reduced Goldilocks Field elements and unreduced field elements. This change would have two positive impacts. First, by limiting the interface for constructing a reduced Goldilocks field element, one may know whether the underlying representation of the Goldilocks field element is reduced from the type alone. This drastically reduces the number of caller assumptions in the codebase. Second, by keeping track programmatically of the maximum number of bits required to represent a given unreduced Goldilocks field elements, reductions may be performed "just-in-time" rather than via manual reasoning.

More detailed reference comments. Since Succinct's code base is almost a direct port of Polygon Zero's plonky2 repository, most of the functions and arguments match the reference implementation almost exactly. However, certain parts of the code are altered or moved across the codebase for efficiency (see, for example, V-SCT-VUL-018). This can lead to some confusion for readers, and may be difficult to keep in sync with changes to the plonky2 repository. We would recommend documenting each deviation from the reference implementation, preferably with a link to the relevant source code. If possible, providing a link to the reference implementation of each function would greatly improve readability, as the plonky2 code repository contains relevant comments and documentation.

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

[§] `https://docs.gnark.consensys.net/HowTo/write/hints`

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
gnark-plonky2-verifier	`0x89b5a01e`	Go	gnark
SuccinctGateway	`0xb1f6bd08` - `0xabd43565`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Nov. 6 - Nov. 27, 2023	Manual & Tools	4	12 person-weeks
Jan. 4 - Jan. 5, 2024	Manual & Tools	2	4 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	3	3
High-Severity Issues	0	0
Medium-Severity Issues	3	3
Low-Severity Issues	9	9
Warning-Severity Issues	12	11
Informational-Severity Issues	12	12
TOTAL	39	38

Table 2.4: Category Breakdown.

Name	Number
Maintainability	13
Logic Error	11
Data Validation	5
Underconstrained Circuit	3
Overconstrained Circuit	2
Locked Funds	1
Denial of Service	1
Access Control	1
Constraint Optimization	1
Gas Optimization	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of gnark-plonky2-verifier's zero-knowledge circuits and the as outlined below. In our audit, we sought to answer the following questions:

- Are any circuit wires under-constrained?
- Can a generated witness deviate from expected behavior?
- Do all parameters have bound checks so that computations do not over/underflow?
- Are there unnecessary constraints that only increase verifier overhead?
- Are Goldilocks Field computations properly emulated within the BN254 field?
- Do the PLONK gates contain arithmetic errors?
- Will the implementation of Fast Reed-Solomon IOPPs accept invalid commitments?
- Does the verifier accept invalid proofs?
- Can any non-permissioned entity execute an unexpected method in the gateway?
- Can the calls and callbacks requested by users be exploited via careful choice of the callback address?
- Are any common Solidity vulnerabilities (reentrancy, large stake holder attacks, unchecked return values, etc.) present in the contracts?
- Can funds be locked in the contracts?

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 tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.
- *Fuzzing/Property-based Testing.* We leverage fuzz testing to determine if the protocol may deviate from the expected behavior (see Chapter 5 for discussion and results). To do this, we formalize the desired behavior of the protocol and then perform differential fuzzing to determine if a violation of the specification can be found.
- *Formal Verification.* We also leverage our custom formal verification tool Picus to verify safety properties of the zero-knowledge circuits (see Chapter 6 for discussion and results). This tool is designed to prove or find violations of determinism, which is an important safety property for zero-knowledge circuits.

Scope. The scope of this audit is limited to the Go language files in the following directories of the repository located at `https://github.com/succinctlabs/gnark-plonky2-verifier` (excluding all `*_utils.go`, `*_test.go`, `*_constants.go`, `util.go`, and `vars.go` files):

- `challenger/`
- `fri/`
- `goldilocks/`
- `plonk/` and `plonk/gates/`
- `poseidon/`
- `verifier/`

For the smart contracts, only the `SuccinctGateway.sol` and `FunctionRegistry.sol` files were in scope.

During the audit, the Veridise auditors referred to the excluded files but assumed that they have been implemented correctly.

Methodology. Veridise auditors inspected the provided tests and read the documentation for both gnark-plonky2-verifier and the SuccinctGateway. They then began a manual audit of the code assisted by both automated testing and verification tools. During the audit, the Veridise auditors regularly met with the gnark-plonky2-verifier developers to ask questions about the code. The Veridise auditors also referred to the original Plonky2 implementation (occasionally cited as "the reference implementation") during the course of the audit to check for differences in gnark-plonky2-verifier's implementation.

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-SCT-VUL-001	inverse missing range check	Critical	Fixed
V-SCT-VUL-002	Unconstrained variable in Reduce()	Critical	Fixed
V-SCT-VUL-003	Unconstrained variable in ReduceWithMaxBits()	Critical	Fixed
V-SCT-VUL-004	MulAdd() may mutate arguments	Medium	Fixed
V-SCT-VUL-005	Missing range-check on EvalProofs	Medium	Fixed
V-SCT-VUL-006	Contracts: funds may be locked when feeVault is. . .	Medium	Fixed
V-SCT-VUL-007	FRI Parameter Ignored During Loading	Low	Fixed
V-SCT-VUL-008	interpolate() over-constrained	Low	Fixed
V-SCT-VUL-009	ArityBits = 8 will provoke errors	Low	Fixed
V-SCT-VUL-010	alpha value overwritten to 1	Low	Fixed
V-SCT-VUL-011	Bitpacking operation may lead to overlap	Low	Fixed
V-SCT-VUL-012	evalL0 overconstrained	Low	Fixed
V-SCT-VUL-013	Salted evaluation not considered	Low	Fixed
V-SCT-VUL-014	Index calculation discrepancy	Low	Fixed
V-SCT-VUL-015	Contracts: _callbackGasLimit is unused	Low	Fixed
V-SCT-VUL-016	1 added to TwoAdicSubgroup output twice	Warning	Fixed
V-SCT-VUL-017	Missed opportunities to use abstractions	Warning	Fixed
V-SCT-VUL-018	Unused functions/variables	Warning	Fixed
V-SCT-VUL-019	Bit reversal incorrect for ArityBits != 4	Warning	Fixed
V-SCT-VUL-020	clearBuffer doesn't clear the buffer	Warning	Fixed
V-SCT-VUL-021	Possible leakage of randomness	Warning	Fixed
V-SCT-VUL-022	Unused challenger parameter due to relocated logic	Warning	Fixed
V-SCT-VUL-023	Errors in gate Id functions	Warning	Fixed
V-SCT-VUL-024	Use of incorrect constant	Warning	Fixed
V-SCT-VUL-025	Missing degree check for CosetInterpolationGate	Warning	Fixed
V-SCT-VUL-026	Contracts: Centralization Risk	Warning	Partially Fixed
V-SCT-VUL-027	Contracts: All _callbackAddresses permitted	Warning	Fixed
V-SCT-VUL-028	Replace GoldilocksHashOut size with named constant	Info	Fixed
V-SCT-VUL-029	Type error in bn254's HashOrNoop()	Info	Fixed
V-SCT-VUL-030	Non-std range-check used	Info	Fixed
V-SCT-VUL-031	Various out-of-date comments and documentation	Info	Fixed
V-SCT-VUL-032	Typos in code	Info	Fixed
V-SCT-VUL-033	Ignored gate parameters	Info	Fixed
V-SCT-VUL-034	Hard-coded constants in code	Info	Fixed
V-SCT-VUL-035	No assertion that gnark is using BN254	Info	Fixed
V-SCT-VUL-036	Inaccurate bounds-check on numConsts	Info	Fixed
V-SCT-VUL-037	Sub-optimal sub-expression in random access gate	Info	Fixed
V-SCT-VUL-038	Contracts: Code Recommendations	Info	Fixed
V-SCT-VUL-039	Contracts: Possible Wasted Gas	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-SCT-VUL-001: inverse missing range check

Severity	Critical	Commit	89b5a01
Type	Underconstrained Circuit	Status	Fixed
File(s)	goldilocks/base.go		
Location(s)	Inverse()		
Confirmed Fix At	9e96393		

The `goldilocks/base.go` file implements standard operations for the Goldilocks field emulated within the base field. One of these operations is `Inverse()`, which computes the inverse of a Goldilocks field element.

The actual computation of the inverse is done using a hint. To ensure correctness, the inverse of `x` is constrained to multiply with `x` to equal 1. This can be seen in the implementation below:

```
func (p *Chip) Inverse(x Variable) Variable {
    result, err := p.api.Compiler().NewHint(InverseHint, 1, x.Limb)
    if err != nil {
        panic(err)
    }

    inverse := NewVariable(result[0])
    product := p.Mul(inverse, x)
    p.api.AssertIsEqual(product.Limb, frontend.Variable(1))
    return inverse
}
```

Snippet 4.1: Implementation of `Inverse()`

However, in the above code, `p.Mul` assumes that `inverse` and `x` are reduced elements of the Goldilocks field. More specifically, `p.Mul(inverse, x)` asserts that there exist field elements `quotient` and `remainder` such that

```
inverse * x == quotient * MODULUS + remainder
0 <= quotient < MODULUS
0 <= remainder < MODULUS
```

Since the `remainder` is returned as the value of the product, the constraints that `Inverse()` encodes can be summarized as follows:

```
inverse * x == quotient * MODULUS + remainder
0 <= quotient < MODULUS
0 <= remainder < MODULUS
remainder == 1
```

Writing `inv(x)` for the inverse of `x` within the base field, we can solve for `inverse`.

```
inverse := inv(x) * (quotient * MODULUS + 1)
```

Since `inverse` is not constrained to be in the Goldilocks field, this is a valid solution for any `0 <= quotient < MODULUS`. Only one of these `MODULUS`-many solutions is the correct reduced inverse.

Impact `Inverse()` may return an element that is not a member of the Goldilocks field and is not equal to the Goldilocks-inverse of `x` when reduced into the field.

To see that the latter statement holds, take any true Goldilocks-inverse `xInv` of `x` with non-zero `quotient`, and note that `xInv - MODULUS` is a valid solution to the `Inverse()` constraints.

Recommendation Perform a range check on `inverse`, requiring it to be a member of the Goldilocks field.

Developer Response We applied the recommended fix.

4.1.2 V-SCT-VUL-002: Unconstrained variable in Reduce()

Severity	Critical	**Commit**	89b5a01
Type	Underconstrained Circuit	**Status**	Fixed
File(s)	`goldilocks/base.go`		
Location(s)	Reduce()		
Confirmed Fix At	297a820		

The `goldilocks/base.go` file emulates the Goldilocks field within the base field. One important operation is `Reduce()`, which maps an element of the base field down to an equivalent (modulo the order of the Goldilocks field) element of the Goldilocks field.

The actual computation of the reduction is done using a hint. However, the result of the hint is never included in a constraint with the variable `x` which is intended to be reduced.

```
1  func (p *Chip) Reduce(x Variable) Variable {
2      // Witness a ‘quotient‘ and ‘remainder‘ such that:
3      //
4      //      MODULUS * quotient + remainder = x
5      //
6      // Must check that offset \in [0, MODULUS) and carry \in [0, 2^
       RANGE_CHECK_NB_BITS) to ensure
7      // that this computation does not overflow. We use 2^RANGE_CHECK_NB_BITS to
       reduce the cost of the range check
8      //
9      // In other words, we assume that we at most compute a a dot product with
       dimension at most RANGE_CHECK_NB_BITS - 128.
10
11     result, err := p.api.Compiler().NewHint(ReduceHint, 2, x.Limb)
12     if err != nil {
13         panic(err)
14     }
15
16     quotient := result[0]
17     p.rangeChecker.Check(quotient, RANGE_CHECK_NB_BITS)
18
19     remainder := NewVariable(result[1])
20     p.RangeCheck(remainder)
21     return remainder
22 }
```

Snippet 4.2: Implementation of `Reduce()`

Since `result` is computed from `x` by a hint, that computation creates no constraints. As a result, any `quotient`, `remainder` pair which satisfy the range checks will satisfy the entire `Reduce()` constraints.

Impact When `Reduce()`ing to the Goldilocks field, an attacker may choose any Goldilocks field element they desire.

Recommendation Require `MODULUS * quotient + remainder` to equal `x`.

Developer Response We applied the recommended fix.

4.1.3 V-SCT-VUL-003: Unconstrained variable in ReduceWithMaxBits()

Severity	Critical	**Commit**	89b5a01
Type	Underconstrained Circuit	**Status**	Fixed
File(s)	goldilocks/base.go		
Location(s)	ReduceWithMaxBits()		
Confirmed Fix At	297a820		

The function `ReduceWithMaxBits()` performs a reduction on a variable `x` which is assumed to be representable in `maxNbBits`. The implementation suffers from the same lack of constraints described in V-SCT-VUL-002 : `x` is not constrained by `ReduceWithMaxBits()`.

```
func (p *Chip) ReduceWithMaxBits(x Variable, maxNbBits uint64) Variable {
    // Witness a ‘quotient‘ and ‘remainder‘ such that:
    //
    //      MODULUS * quotient + remainder = x
    //
    // Must check that remainder \in [0, MODULUS) and quotient \in [0, 2^maxNbBits)
    to ensure that this
    // computation does not overflow.

    result, err := p.api.Compiler().NewHint(ReduceHint, 2, x.Limb)
    if err != nil {
        panic(err)
    }

    quotient := result[0]
    p.rangeChecker.Check(quotient, int(maxNbBits))

    remainder := NewVariable(result[1])
    p.RangeCheck(remainder)
    return remainder
}
```

Snippet 4.3: Implementation of `ReduceWithMaxBits()`

Since `result` is computed from `x` by a hint, that computation creates no constraints. As a result, any `quotient`, `remainder` pair which satisfy the range checks will satisfy the entire `ReduceWithMaxBits()` constraints.

Impact When calling `ReduceWithMaxBits()` to map an element into the Goldilocks field, an attacker may choose any Goldilocks field element they desire.

Recommendation Require `MODULUS * quotient + remainder` to equal `x`.

Developer Response We applied the recommended fix.

4.1.4 V-SCT-VUL-004: MulAdd() may mutate arguments

Severity	Medium	**Commit**	89b5a01
Type	Logic Error	**Status**	Fixed
File(s)	`goldilocks/base.go`		
Location(s)	MulAddNoReduce(), MulAdd()		
Confirmed Fix At	a6707ed		

The documentation for `frontend.API.MulAcc` states that it *may* modify the first argument:

```
// MulAcc sets and return a = a + (b*c).
//
// ! The method may mutate a without allocating a new result. If the input
// is used elsewhere, then first initialize new variable, for example by
// doing:
//
//     acopy := api.Mul(a, 1)
//     acopy = MulAcc(acopy, b, c)
//
// ! But it may not modify a, always use MulAcc(...) result for correctness.
MulAcc(a, b, c Variable) Variable
```

Snippet 4.4: Documentation of `frontend.API.MulAcc`

Both `MulAdd` and `MulAccNoReduce` in `goldilocks/base.go` call `MulAcc` directly on `c.Limb` (from argument `c`). For example, see the below definition of `MulAddNoReduce()`.

```
func (p *Chip) MulAddNoReduce(a Variable, b Variable, c Variable) Variable {
        return NewVariable(p.api.MulAcc(c.Limb, a.Limb, b.Limb))
}
```

Snippet 4.5: Definition of `MulAddNoReduce()`

Impact If the `c.Limb` `frontend.Variable` is a pointer type (which is allowed and does occur in this codebase), it is possible for `MulAdd` and `MulAddNoReduce` to mutate their arguments and propagate unintended modifications to other callsites. This potentially affects many locations, since `MulAdd` and `MulAddNoReduce` are the base operations of many other functions, including `Add` and `Sub` in `goldilocks/base.go`, `AddExtension` and `SubExtension` in `goldilocks/quadratic_extension.go`, and `AddExtensionAlgebra` and `SubExtensionAlgebra` in `goldilocks/quadratic_extension_algebra.go`. Some possible affected callsites where `MulAcc` may affect subsequent computation include:

- `evalVanishingPoly` in `plonk/plonk.go`: `wireValuePlusGamma` may be modified and taint the `denominator` calculation.
- `EvalUnfiltered` in `plonk/gates/poseidon_gate.go`: `deltaI` may be modified and taint the `state[i+4]` calculation.
- `friCombineInitial` in `fri/fri.go`: `subgroupX_QE` may be modified.
- `interpolate` in `fri/fri.go`: `x` may be modified for future loop iterations.
- `EvalUnfiltered` in `plonk/gates/constant_gate.go`: `localConstants` may be modified.
- `EvalUnfiltered` in `plonk/gates/public_input_gate.go`: `localWires` may be modified.

Recommendation Follow the recommendation in the gnark documentation and copy `c.Limb` before performing the `MulAcc` operation for safety:

```
cLimbCopy := p.api.Mul(c.Limb, 1)
p.api.MulAcc(cLimbCopy, a.Limb, b.Limb)
```

Developer Response We now make a copy of the first argument before passing it to `frontend.api.MulAcc` in all cases where this could be an issue.

4.1.5 V-SCT-VUL-005: Missing range-check on EvalProofs

Severity	Medium	**Commit**	89b5a01
Type	Data Validation	**Status**	Fixed
File(s)	`verifier/verifier.go`		
Location(s)	rangeCheckProof()		
Confirmed Fix At	6af5b0a		

Only the first item of the `InitialTreesProof.EvalProofs` list is range checked (i.e., only on `queryRound.InitialTreesProof.EvalsProofs[0]`):

```
// Range check the openings proof.
for _, queryRound := range proof.OpeningProof.QueryRoundProofs {
      for _, initialTreesElement := range queryRound.InitialTreesProof.EvalsProofs
    [0].Elements {
        c.glChip.RangeCheck(initialTreesElement)
      }


      for _, queryStep := range queryRound.Steps {
          for _, eval := range queryStep.Evals {
                  c.glChip.RangeCheckQE(eval)
          }
      }
}
```

Snippet 4.6: Range check performed on the openings proof.

Impact The values of `EvalsProofs` are used in `fri/fri.go` (in `verifyInitialProof` and in `friCombineInitial`), so the non-range-checked inputs could result in the violation of other assumptions in the codebase.

For example, the unchecked `Elements` are eventually hashed in `BN254Chip.HashNoPad` (`verifyInitialProof` → `verifyMerkleProofToCapWithCapIndex` → `HashOrNoop` → `HashNoPad`), which assumes that the inputs are reduced goldilocks field elements (as the bit-packing operation assumes the inputs are at most 64 bits).

Recommendation Perform range checks on all elements of the `EvalsProofs` list.

Developer Response We applied the recommended fix.

4.1.6 V-SCT-VUL-006: Contracts: funds may be locked when feeVault is disabled

Severity	Medium	Commit	abd4356
Type	Locked Funds	Status	Fixed
File(s)	contracts/src/SuccinctGateway.sol		
Location(s)	SuccinctGateway		
Confirmed Fix At	https://github.com/succinctlabs/succinctx/pull/312		

The `SuccinctGateway` contract allows a privileged user (called the guardian) to set the address of the `IFeeVault` contract that will hold fees paid by users for their requests. This is done via the `setFeeVault()` function.

```
/// @dev Sets the fee vault to a new address. Can be set to address(0) to disable
    fees.
/// @param _feeVault The address of the fee vault.
function setFeeVault(address _feeVault) external onlyGuardian {
    emit SetFeeVault(feeVault, _feeVault);
    feeVault = _feeVault;
}
```

Snippet 4.7: Definition of `SuccinctGateway.setFeeVault()`

The documentation states that the fee vault address "can be set to address(0) to disable fees" and both the `requestCall()` and `requestCallback()` functions correctly ensure `feeVault != address(0)` before depositing the `msg.value` into the `feeVault` contract. However, when `feeVault == address(0)` the `msg.value` funds sent by `msg.sender` with the transaction are retained in the `SuccinctGateway` contract with no way for the user to retrieve them.

Impact If `feeVault == address(0)` and a user sends funds when calling the `requestCall()` or `requestCallback()` functions, those funds will be locked in the `SuccinctGateway` contract.

Recommendation The `SuccinctGateway` contract should provide a way for the user to retrieve their funds that were locked because `feeVault == address(0)`.

Developer Response We added a `recover()` function to the SuccinctGateway to allow the guardian to initiate a fund transfer to a given address.

4.1.7 V-SCT-VUL-007: FRI Parameter Ignored During Loading

Severity	Low	**Commit**	89b5a01
Type	Logic Error	**Status**	Fixed
File(s)	`types/common_data.go`		
Location(s)	ReadCommonCircuitData()		
Confirmed Fix At	f71795a		

Note: This is technically out of scope, but our team noticed this issue while reviewing the type representation of the different elements inside the codebase.

Function `ReadCommonCircuitData` populates all fields for an instance of type `CommonCircuitData`, which is then returned to the caller. However, when populating the `FriParams` (see snippet bellow), parameter `Hiding` is ignored.

```
commonCircuitData.FriParams.Config.RateBits = raw.FriParams.Config.RateBits
commonCircuitData.FriParams.Config.CapHeight = raw.FriParams.Config.CapHeight
commonCircuitData.FriParams.Config.ProofOfWorkBits = raw.FriParams.Config.
    ProofOfWorkBits
commonCircuitData.FriParams.Config.NumQueryRounds = raw.FriParams.Config.
    NumQueryRounds
commonCircuitData.FriParams.ReductionArityBits = raw.FriParams.ReductionArityBits
```

Snippet 4.8: Snippet from `ReadCommonCircuitData`

Impact Function `validateFriProofShape` might panic for proofs whose `Hiding` parameter is set to true. This can negatively impact some users when trying to generate a proof.

Recommendation We recommend loading the `Hiding` parameter in function `ReadCommonCircuitData` .

Developer Response We have decided to require the `Hiding` parameter to be set to `false`.

4.1.8 V-SCT-VUL-008: interpolate() over-constrained

Severity	Low	**Commit**	89b5a01
Type	Overconstrained Circuit	**Status**	Fixed
File(s)	`fri/fri.go`		
Location(s)	interpolate()		
Confirmed Fix At	d241f54		

The `interpolate()` function takes a vector of `xPoints` and `yPoints` as inputs and uses them to evaluate the polynomial they define at a new point `x`. This is handled in two cases:

1. When `x` is not equal to any `xPoints[i]`.
2. When `x` is equal to some `xPoints[i]`.

In case 1, a division with a denominator of `x - xPoints[i]` is computed.

```
f.gl.DivExtension(
    barycentricWeights[i],
    f.gl.SubExtension(
        x,
        xPoints[i],
    ),
),
```

Snippet 4.9: Division by `x - xPoints[i]` in `interpolate()`.

Case 2, later on in the function, uses selectors to evaluate to `yPoints[i]` if `x = xPoints[i]`.

However, `DivExtension` computes the `Inverse` directly, requiring it to exist.

```
func (p *Chip) InverseExtension(a QuadraticExtensionVariable)
    QuadraticExtensionVariable {
    a0IsZero := p.api.IsZero(a[0].Limb)
    a1IsZero := p.api.IsZero(a[1].Limb)
    p.api.AssertIsEqual(p.api.Mul(a0IsZero, a1IsZero), frontend.Variable(0))
```

Snippet 4.10: The beginning of `InverseExtension()`, which asserts the existence of an inverse.

So, whenever case 2 occurs, the constraints for case 1 will be unsatisfiable.

Impact Evaluating the polynomial at one of the `xPoints` using `interpolate()` results in unsatisfiable constraints.

Recommendation To make case 2 satisfiable, the developers need to perform an unsafe division, and then use the `yPoints` whenever the denominator is zero.

See also V-SCT-VUL-012.

Developer Response [If applicable]

4.1.9 V-SCT-VUL-009: ArityBits = 8 will provoke errors

Severity	Low	**Commit**	89b5a01
Type	Logic Error	**Status**	Fixed
File(s)	`fri/fri.go`		
Location(s)	computeEvaluation()		
Confirmed Fix At	c0cbac8		

The `computeEvaluation()` function uses an `arityBits` of at most 8 to evaluate the next reduced FRI polynomial. `evals`, which is used to perform this computation, is an array of length `arity := 2^arityBits`.

```
func (f *Chip) computeEvaluation(
    // [VERIDISE] ...
) gl.QuadraticExtensionVariable {
    arity := 1 << arityBits
    if (len(evals)) != arity {
        panic("len(evals) ! arity")
    }
    if arityBits > 8 {
        panic("currently assuming that arityBits is <= 8")
    }
```

Snippet 4.11: Checks at the beginning of `computeEvaluation()`

Later, when permuting the evaluations, 8-bit indices are used.

```
permutedEvals := make([]gl.QuadraticExtensionVariable, len(evals))
for i := uint8(0); i < uint8(len(evals)); i++ {
    newIndex := bits.Reverse8(i) >> arityBits
    permutedEvals[newIndex] = evals[i]
}
```

Snippet 4.12: Evaluation of `permutedEvals`.

Casting `len(evals)` to `uint8` will truncate any values larger than 255. If `arityBits == 8`, then `len(evals) == 256`, and `uint8(len(evals))` evaluates to zero.

Impact For an `arityBits` of 8, this function will cause errors due to invalid memory accesses. Currently, the only callers use `arityBits == 4`, so this will only be an issue in future iterations of the protocol.

Recommendation Require `arityBits` to be at most 7, or handle the `arityBits == 8` case separately.

Developer Response We now use a <= bounds-check with `uint8(len(evals)-1)`.

4.1.10 V-SCT-VUL-010: alpha value overwritten to 1

Severity	Low	**Commit**	89b5a01
Type	Logic Error	**Status**	Fixed
File(s)	`poseidon/bn254.go`		
Location(s)	HashOrNoop()		
Confirmed Fix At	5766879		

The `big.Int.Exp` function documentation states that, for `z.Exp(x, y, m)`:

> Exp sets z = x**y mod |m| (i.e. the sign of m is ignored), and returns z.

So, in the loop body:

```
for i, inputElement := range input {
    returnVal = c.api.MulAcc(returnVal, inputElement, alpha.Exp(alpha, big.NewInt(
    int64(i)), nil))
}
```

Snippet 4.13: Loop body in `HashOrNoop`.

During the execution of `alpha.Exp(alpha, big.NewInt(int64(i)), nil)`, `alpha` will be set to 1 (as `i = 0` in the first iteration) and will remain 1 for all future iterations.

Impact This changes the computation of the function to effectively `sum(input)`, which does not appear to be the desired computation.

Recommendation Change `alpha.Exp(...)` to `new(big.Int).Exp(...)`, or remove this logic if unused.

Developer Response We applied the recommended fix.

4.1.11 V-SCT-VUL-011: Bitpacking operation may lead to overlap

Severity	Low	Commit	89b5a01
Type	Logic Error	Status	Fixed
File(s)	`poseidon/bn254.go`		
Location(s)	HashOrNoop()		
Confirmed Fix At	5766879		

In `HashOrNoop`, if there are fewer than 4 elements in the input vector (`len(input) <= 3`), the function attempts to pack the elements into a single BN254 variable and return the packed value otherwise unmodified:

```
if len(input) <= 3 {
        returnVal := frontend.Variable(0)

    alpha := new(big.Int).SetInt64(1 << 32)
    for i, inputElement := range input {
        returnVal = c.api.MulAcc(returnVal, inputElement, alpha.Exp(alpha, big.NewInt
    (int64(i)), nil))
    }
    return BN254HashOut(returnVal)
} else { ... }
```

Snippet 4.14: Case in `HashOrNoop` when `len(input) <= 3`.

(Note that this does not function correctly due to the prior issue V-SCT-VUL-010).

The value of `alpha` is `1 << 32`; however, Goldilocks variables may take up 64 bits, and so the packing operation may end up modifying some of the bits of each of the input elements while constructing the output value. This logic appears to be implemented correctly in `HashNoPad`, which performs this operation with `two_to_64`:

```
two_to_32 := new(big.Int).SetInt64(1 << 32)
two_to_64 := new(big.Int).Mul(two_to_32, two_to_32)
...
inter := frontend.Variable(0)
for k := 0; k < len(bn254Chunk); k++ {
        inter = c.api.MulAcc(inter, bn254Chunk[k].Limb, new(big.Int).Exp(two_to_64,
    big.NewInt(int64(k)), nil))
}
```

Snippet 4.15: Bitpacking operation in `HashNoPad`.

Impact This changes the computation of the function, as the function does perform computation over the input values and is not a no-op as the function name implies.

Recommendation Change alpha to `1 << 64` (see `HashNoPad` for computation of `two_to_64`), or remove this logic if unused.

Developer Response We applied the recommended fix by replacing `alpha := (1 << 32)` with `alpha := (1 << 32)^2`.

4.1.12 V-SCT-VUL-012: evalL0 overconstrained

Severity	Low	**Commit**	89b5a01
Type	Overconstrained Circuit	**Status**	Fixed
File(s)	`plonk/plonk.go`		
Location(s)	evalL0()		
Confirmed Fix At	d241f54		

The `evalL0` function evaluates `(x^n - 1) / (n * (x-1))`.

```
denominator := glApi.SubExtension(
    glApi.ScalarMulExtension(x, p.DEGREE),
    p.DEGREE_QE,
)
return glApi.DivExtension(
    evalZeroPoly,
    denominator,
)
```

Snippet 4.16: Computation of and division by `n * (x - 1)`, computed as `n * x - n`.

As described in V-SCT-VUL-008, `DivExtension()` is a *safe* division, asserting that `denominator != 0`. If `x` happens to equal 1, then the denominator will be zero, and these constraints will be unsatisfiable.

Note that 0xPolygonZero's plonky2 repository handles the `x = 1` case by returning 1 directly.

Impact If the evaluation point ever happens to be 1, the circuits will become unsatisfiable. In this case, the prover will need to generate a new proof, likely going through the grinding process again to generate new challenges.

Recommendation As the uses of `DivExtension()` in this issue and in V-SCT-VUL-008 are the only (non-test) uses of `DivExtension()`, consider making a function `DivExtensionOr()`, which returns the division when the denominator is non-zero and a default value otherwise.

Developer Response [If applicable]

4.1.13 V-SCT-VUL-013: Salted evaluation not considered

Severity	Low	**Commit**	89b5a01
Type	Logic Error	**Status**	Fixed
File(s)	`fri/fri.go`		
Location(s)	friCombineInitial()		
Confirmed Fix At			

In `friCombineInitial()`, the oracle evaluations of the polynomials from each batch are gathered in a loop.

```
for i := 0; i < len(instance.Batches); i++ {
        batch := instance.Batches[i]
        reducedOpenings := precomputedReducedEval[i]

        point := batch.Point
        evals := make([]gl.QuadraticExtensionVariable, 0)
        for _, polynomial := range batch.Polynomials {
            evals = append(
                evals,
                gl.QuadraticExtensionVariable{
                    proof.EvalsProofs[polynomial.OracleIndex].Elements[polynomial.
    PolynomialInfo],
                    gl.Zero(),
                },
            )
        }
```

Snippet 4.17: Snippet from `friCombineInitial()`.

In the corresponding location in the plonky2 codebase, a variable `salted` is computed and the evaluation is computed using `unsalted_eval()`.

```
let poly_blinding = instance.oracles[p.oracle_index].blinding;
let salted = params.hiding && poly_blinding;
proof.unsalted_eval(p.oracle_index, p.polynomial_index, salted)
```

Snippet 4.18: Computation of each element of `evals` in the plonky2 repository.

The definition of `unsalted_eval()` is shown below. `salt_size(b)` is equivalent to the statement `if b SALT_SIZE else 0`.

```
pub(crate) fn unsalted_eval(&self, oracle_index: usize, poly_index: usize, salted:
    bool) -> F {
    self.unsalted_evals(oracle_index, salted)[poly_index]
}

fn unsalted_evals(&self, oracle_index: usize, salted: bool) -> &[F] {
    let evals = &self.evals_proofs[oracle_index].0;
    &evals[..evals.len() - salt_size(salted)]
}
```

Snippet 4.19: Definition of `unsalted_eval()`

As can be seen, the Go implementation assumes `salted` is set to `false`. Since each plonk oracle in the Go repository does have a `Blinding` parameter which may be `true` or `false`, this equates to hard-coding `params.Hiding` to `false`.

However, the `params.Hiding` attribute is used elsewhere in the codebase, e.g. in `validateFriProofShape()`, so it is not assumed to be `false` everywhere.

Impact Extra work may occur to validate the salt evaluations. A prover working based off of the plonky2 repository may incorrectly elide these evaluations.

Recommendation Implement the logic of `unsalted_eval()`.

Developer Response We have decided to require the `Hiding` parameter to be set to `false`.

4.1.14 V-SCT-VUL-014: Index calculation differs from reference implementation

Severity	Low	**Commit**	89b5a01
Type	Logic Error	**Status**	Fixed
File(s)	`plonk/gates/coset_interpolation_gate.go`		
Location(s)	EvalUnfiltered()		
Confirmed Fix At	5dd6da2		

The calculation of `endIndex` in function `EvalUnfiltered` of the `CosetInterpolationGate` deviates from the reference implementation. Specifically, the reference implementation takes the min between `start_index + degree - 1` and `num_points` (see here). The `gnark` implementation, on the other hand, does not consider the number of points at all (see snippet bellow).

```
endIndex := startIndex + g.degree - 1
```

Snippet 4.20: Snippet from `EvalUnfiltered()`.

Impact This may result in a crash due to out of bounds index or, worse, wrong constraint evaluation.

Recommendation We recommend making the `gnark` implementation consistent with the reference one.

Developer Response We applied the recommended fix.

4.1.15 V-SCT-VUL-015: Contracts: _callbackGasLimit is unused

Severity	Low	**Commit**	abd4356
Type	Denial of Service	**Status**	Fixed
File(s)	`contracts/src/SuccinctGateway.sol`		
Location(s)	fulfillCallback()		
Confirmed Fix At	`https://github.com/succinctlabs/succinctx/pull/316`		

The `SuccinctGateway` contract contains a `fulfillCallback()` function that the prover calls to fulfill an earlier request made via the `requestCallback()` function.

```
/// @dev Fulfills a request by providing the output and proof.
/// @param _nonce The nonce of the request.
/// @param _functionId The function identifier.
/// @param _inputHash The hash of the function input.
/// @param _callbackAddress The address of the callback contract.
/// @param _callbackSelector The selector of the callback function.
/// @param _callbackGasLimit The gas limit for the callback function.
/// @param _context The function context.
/// @param _output The function output.
/// @param _proof The function proof.
function fulfillCallback(
```

Snippet 4.21: Definition of `SuccinctGateway.fulfillCallback()`

The `_callbackGasLimit` parameter of the `fulfillCallback()` function is not used within that function to limit the gas usage of the callback function.

Impact The callback function can contain arbitrary code that may have a very high gas usage or exhaust the gas limit altogether and cause the callback to revert. If the provers are expected to pay the gas fees, a malicious callback function may attack their solvency.

Recommendation If the intention is for the user to pay the callback gas fees, and for the provers to pay the fees of the `fulfillCallback()` function itself, the callback gas should be limited by using `.call{gas: _callbackGasLimit}(...)`.

Note that, in this case, there is a still the opportunity for a "returnbomb," in which the user callback returns a very large value. Though the `fulfillCallback()` function does not use the return value, the generated EVM code will copy the entire returndata to memory after the call. To mitigate this, inline assembly can be used instead of the Solidity `call()` function so that the return data can be ignored altogether. See `https://github.com/nomad-xyz/ExcessivelySafeCall` for a more complete description and solution.

Developer Response Although we generally will always simulate before sending for `fulfill*()` transactions, we added `{gas: _callbackGasLimit}` to the `_callbackAddress.call()` statement as an extra precaution.

4.1.16 V-SCT-VUL-016: 1 added to TwoAdicSubgroup output twice

Severity	Warning	**Commit**	89b5a01
Type	Logic Error	**Status**	Fixed
File(s)	`goldilocks/base.go`		
Location(s)	TwoAdicSubgroup()		
Confirmed Fix At	2043890		

1 is added to the output vector `res` before the `for` loop, which then computes the remainder of the subgroup with the primitive root of unity (i.e., $\omega, \omega^2, \ldots$).

```
res = append(res, goldilocks.NewElement(1))

for i := 0; i < (1 << nLog); i++ {
      lastElement := res[len(res)-1]
        res = append(res, *lastElement.Mul(&lastElement, &rootOfUnity))
}
```

Snippet 4.22: Loop computing the two-adic subgroup in `TwoAdicSubgroup`.

However, the loop iterates through values $[0, 2^{nLog} - 1]$, which compute output values ranging from $[\omega, \omega^{2^{nLog}}]$. Since $\omega^{2^{nLog}} = 1$, the last element added to the `res` vector will be 1, which is redundant, as 1 was already added to the vector.

Impact The output vector will contain the element 1 twice (i.e., $[1, \omega, \omega^2, \ldots, w^{2^{nLog-1}}, 1]$).

Recommendation Change the loop iteration to be bounded on `i < (1 << nLog) - 1`.

Developer Response We applied the recommended fix.

4.1.17 V-SCT-VUL-017: Missed opportunities to use appropriate abstractions

Severity	Warning	**Commit**	89b5a01
Type	Maintainability	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	9617141		

The following locations use inlined expressions instead of their already-created abstractions:

- `challenger/challenger.go`:
  - The declaration of `spongeState [poseidon.SPONGE_WIDTH]gl.Variable` can be replaced with `spongeState gl.GoldilocksState`.
- `goldilocks/base.go`:
  - Functions `Sub` and `SubNoReduce` can replace `MODULUS.Uint64()-1` with `NegOne()`.
  - Function `Reduce` duplicates the logic of `ReduceWithMaxBits`, so the body of `Reduce` could be replaced with a single call: `ReduceWithMaxBits(x, RANGE_CHECK_NB_BITS)`.
- `goldilocks/quadratic_extension.go`:
  - Function `MulExtensionNoReduce` could have replaced `7` with `W`.
  - Function `InverseExtension` could have replaced `p.api.IsZero(...Limb)` with `p.IsZero(...)`.
  - Functions `MulAddExtension`, `MulExtension`, `SubMulExtension` could have replaced two calls to `Reduce` with a single call to `ReduceExtension`.
  - Functions `AddExtension` and `SubExtension` could replace their calls to `Add/Sub` to `AddExtensionNoReduce/SubExtensionNoReduce` followed by a call to `ReduceExtension`.
- `goldilocks/quadratic_extension_algebra.go`:
  - Function `ToQuadraticExtensionAlgebra` could have replaced `2` with `D`.
- `poseidon/goldilocks.go`:
  - Function `HashNoPad` could have replaced number `4` with `len(hash)`.
  - Function `constantLayer` could have replaced `c.gl.MulAdd(..., gl.NewVariable(1), ...)` with `c.gl.Add(..., ...)`.
  - Functions `mdsRowShf` and `MdsRowShfExtension` could have replaced `[SPONGE_WIDTH]gl.Variable` with `GoldilocksState`.
  - Function `mdsPartialLayerInit` could replace `gl.NewVariable(0)` with a call to `gl.Zero()`.
- `plonk/gates/reducing_gate.go`:
  - In function `EvalUnfiltered`, `coeff` could be initialized with `ToQuadraticExtensionAlgebra(ToQuadraticExtension(NewVariable(coeffs[i])))` rather than looping over `D`. A similar issue occurs in numerous locations.

Impact This inlining makes the code slightly more verbose, and if the internal structure changes (e.g. from refactoring), it would be easier and less error-prone to make one change to the abstraction rather than at all inlined sites.

Recommendation Use the appropriate abstractions.

Developer Response We applied the recommended fix.

4.1.18 V-SCT-VUL-018: Unused functions/variables

Severity	Warning	**Commit**	89b5a01
Type	Maintainability	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	1a03726		

A few functions and variables are defined, but not used in the codebase.

- `goldilocks/base.go`:
  - `var REDUCE_NB_BITS_THRESHOLD uint8` is unused.
  - `func (p *Chip) Exp(...)` is unused.
- `poseidon/goldilocks.go`:
  - `const MAX_WIDTH` is unused.
- `types/utils.go`:
  - `func ReductionArityBits()` is unused.
- `variables/plonk.go`
  - `func NewOpeningSet(...)` is unused.

Impact If unmaintained, these functions may become out-of-sync with the codebase and produce issues down the road if used.

Recommendation Remove the unused values.

Developer Response We applied the recommended fix.

4.1.19 V-SCT-VUL-019: Bit reversal incorrect for ArityBits != 4

Severity	Warning	**Commit**	89b5a01
Type	Logic Error	**Status**	Fixed
File(s)	`fri/fri.go`		
Location(s)	computeEvaluation()		
Confirmed Fix At	2fab6a9		

The `computeEvaluation()` function uses an `arityBits` of at most 8 to evaluate the next reduced FRI polynomial. `evals`, which is used to perform this computation, is an array of length `arity := 2^arityBits`. When permuting the evaluations, 8-bit indices are used.

```
permutedEvals := make([]gl.QuadraticExtensionVariable, len(evals))
for i := uint8(0); i < uint8(len(evals)); i++ {
    newIndex := bits.Reverse8(i) >> arityBits
    permutedEvals[newIndex] = evals[i]
}
```

Snippet 4.23: Evaluation of `permutedEvals`.

The reversed bits should be shifted by `8 - arityBits`, not `arityBits`. For example, if `arityBits` is 8, this will always return 0.

Impact For `arityBits != 4`, the wrong indices will be used. Currently, the only callers use `arityBits == 4`, so this will only be an issue in future iterations of the protocol.

Recommendation Shift by `8 - arityBits` rather than `arityBits`.

Developer Response We applied the recommended fix.

4.1.20 V-SCT-VUL-020: clearBuffer doesn't clear the buffer

Severity	Warning	**Commit**	89b5a01
Type	Logic Error	**Status**	Fixed
File(s)	`challenger/challenger.go`		
Location(s)	clearBuffer()		
Confirmed Fix At	0f6466c		

Function `clearBuffer` does not perform the expected operation of clearing a provided buffer; rather, it ignores its input argument and instead returns a new buffer.

```
1 func clearBuffer(buffer []gl.Variable) []gl.Variable {
2     return make([]gl.Variable, 0)
3 }
```

Snippet 4.24: Definition of function `clearBuffer`.

Impact This has the potential to cause programming errors and confusion about how and when the buffers are cleared.

See related issue V-SCT-VUL-021.

Recommendation Either fix the implementation of `clearBuffer` so that it clears the buffer provided as an argument or rename the function to `newBuffer` and remove the unused parameter.

Developer Response We removed the `clearBuffer()` function to increase the code clarity.

4.1.21 V-SCT-VUL-021: Possible leakage of randomness

Severity	Warning	**Commit**	89b5a01
Type	Logic Error	**Status**	Fixed
File(s)	challenger/challenger.go		
Location(s)	duplexing()		
Confirmed Fix At	0f6466c		

The incorrect behavior of the `clearBuffer` function (see V-SCT-VUL-020) leads to an unintended consequence in the `duplexing` function, which leaves the `outputBuffer` unmodified when it was intended to be cleared before adding new data:

```
clearBuffer(c.outputBuffer)
for i := 0; i < poseidon.SPONGE_RATE; i++ {
        c.outputBuffer = append(c.outputBuffer, c.spongeState[i])
}
```

Snippet 4.25: Operations on `outputBuffer` in the `duplexing` function.

In the current implementation, the `outputBuffer` will already be empty here due to the proper clearing operation performed in `ObserveElement` (`c.outputBuffer = clearBuffer(c.outputBuffer)`).

Impact If the API were to change or the implementation of `ObserveElement` were to change, the `outputBuffer` wouldn't be properly cleared in the `duplexing` function. This could lead to an adaptive attack. The next challenge value would not be dependent on the most recent input, and an attacker could leverage knowledge of the previous outputs to construct proof values that would pass without being an otherwise legitimate proof.

Recommendation Add an assertion into the `duplexing` function that ensures that the `outputBuffer` is empty before appending newly computed values to ensure Fiat-Shamir is performed safely.

Developer Response We applied the recommended fix.

4.1.22 V-SCT-VUL-022: Unused challenger parameter due to relocated logic

Severity	Warning	**Commit**	89b5a01
Type	Maintainability	**Status**	Fixed
File(s)	`challenger/challenger.go`		
Location(s)	GetFriChallenges()		
Confirmed Fix At	30d73da		

In the function `GetFriChallenges`, the parameter `degreeBits` is unused. This is because the FRI query index computation has been moved into the `verifyQueryRound` function in `fri/fri.go`.

```
xIndex = f.gl.Reduce(xIndex)
xIndexBits := f.api.ToBinary(xIndex.Limb, 64)[0 : f.friParams.DegreeBits+f.friParams.
    Config.RateBits]
```

Snippet 4.26: FRI index computation.

This is a change from the original plonky2 implementation, which performs the computation in the `fri_challenges` function (the analogue of the `GetFriChallenges` function).

Impact This could cause confusion for developers and lead to the introduction of redundant computation if someone believes this computation has been erroneously omitted.

Recommendation Remove the unused parameter and add documentation explaining where the logic is performed instead.

Developer Response We applied the recommended fix.

4.1.23 V-SCT-VUL-023: Errors in gate Id functions

Severity	Warning	**Commit**	89b5a01
Type	Data Validation	**Status**	Fixed
File(s)	See issue description		
Location(s)	Id()		
Confirmed Fix At	cc064ae		

The following gates generate the incorrect Id with their `Id` functions:

- `base_sum_gate.go`:
  - Parameter should be `num_limbs` not `num_ops`.
- `multiplication_extension_gate.go`:
  - Id is listed as `ArithmeticExtensionGate` instead of `MulExtensionGate`.
- `reducing_extension_gate.go`:
  - Parameter should be `num_coeffs` not `num_ops`.
- `reducing_gate.go`:
  - Id is listed as `ReducingExtensionGate` instead of `ReducingGate`.
  - Parameter should be `num_coeffs` not `num_ops`.

Impact These errors result in gate Ids being misreported as the incorrect gate or a gate with mismatched parameters.

Recommendation Ensure that gate Id functions are up to date.

Developer Response We applied the recommended fix.

4.1.24 V-SCT-VUL-024: Use of incorrect constant

Severity	Warning	**Commit**	89b5a01
Type	Maintainability	**Status**	Fixed
File(s)	`fri/fri.go`		
Location(s)	verifyMerkleProofToCapWithCapIndex()		
Confirmed Fix At	318c3ce		

At the end of `verifyMerkleProofToCapWithCapIndex()`, a series of lookups are used to identify the correct Merkle cap.

```
// We assume that the cap_height is 4.  Create two levels of the Lookup2 circuit
if len(capIndexBits) != 4 || len(merkleCap) != 16 {
    // [VERIDISE] ... error message here
}

const NUM_LEAF_LOOKUPS = 4
var leafLookups [NUM_LEAF_LOOKUPS]poseidon.BN254HashOut
// First create the "leaf" lookup2 circuits
// The will use the least significant bits of the capIndexBits array
for i := 0; i < NUM_LEAF_LOOKUPS; i++ {
    leafLookups[i] = f.api.Lookup2(
        capIndexBits[0], capIndexBits[1],
        merkleCap[i*NUM_LEAF_LOOKUPS], merkleCap[i*NUM_LEAF_LOOKUPS+1], merkleCap[i*
    NUM_LEAF_LOOKUPS+2], merkleCap[i*NUM_LEAF_LOOKUPS+3],
    )
}
```

Snippet 4.27: Lookup assumptions and first round of lookups to find the correct Merkle cap.

Note the arguments passed to `Lookup2` are `merkleCap[i*NUM_LEAF_LOOKUPS]` through `merkleCap[i*NUM_LEAF_LOOKUPS+3]`.

However, the loop stride over `merkleCap` is a fixed constant determined by `Lookup2`: 4. If `NUM_LEAF_LOOKUPS` changes to a non-4 value, this loop will be incorrect.

Impact If `len(merkleCaps)` were some other power of 4 (e.g. 4 or 64), then `NUM_LEAF_LOOKUPS` would also need to be changed (e.g. to 1 or 16). In this case, the parameters passed to `Lookup2()` would be incorrect.

For example, if `len(merkleCaps)` were changed to 64, then the indexing operations would provoke an out-of-bounds error.

Recommendation Access `merkleCap` on the range `4*i..4*i+3`.

Developer Response We applied the recommended fix, using a new constant `STRIDE_LENGTH = 4`.

4.1.25 V-SCT-VUL-025: Missing degree check for CosetInterpolationGate

Severity	Warning	Commit	89b5a01
Type	Data Validation	Status	Fixed
File(s)	plonk/gates/coset_interpolation_gate.go		
Location(s)	deserializeCosetInterpolationGate()		
Confirmed Fix At	888b247		

In `deserializeCosetInterpolationGate`, the parsed value for `degree` is not checked to be within a certain range. However, according to the reference implementation, the degree needs to be at least 2 in order for the coset interpolation to function properly.

Impact The missing check could lead to a divide-by-zero error in `numIntermediates()` if `g.degree == 1` or invalid computation if `g.degree == 0`.

Recommendation Add an assertion to check that `degreeInt > 1` in `deserializeCosetInterpolationGate` .

Developer Response We applied the recommended fix.

4.1.26 V-SCT-VUL-026: Contracts: Centralization Risk

Severity	Warning	**Commit**	abd4356
Type	Access Control	**Status**	Partially Fixed
File(s)	`contracts/src/SuccinctGateway.sol`		
Location(s)	SuccinctGateway		
Confirmed Fix At	`https://github.com/succinctlabs/succinctx/pull/310`		

The `SuccinctGateway` contract has a special set of accounts (called `guardians`) which can perform privileged actions. This includes two primary actions:

1. Setting the `feeVault` address.
2. Adding to/removing from the whitelisted `allowedProvers` list.

A second privileged role (called the `TIMELOCK_ROLE`) is given the ability to upgrade the contract.

Impact While the centralization risk associated to the `TIMELOCK_ROLE` is reduced by assigning the role to a time-locked contract, if any `guardian`'s keys are compromised users may be harmed in the following ways:

1. The fee vault address may be changed, stealing funds from the manager of the contract.
2. A malicious prover may use the `fulfillCall()` functions to make arbitrary calls on the contract's behalf.

Recommendation To mitigate the risks above, developers should take actions such as:

- Always use the `SuccinctGateway` contract with a multi-signature wallet acting as its owner.
- Actively monitor the current `guardian` set.
- Introduce a time delay before key operations are performed (such as changing the `allowedProvers`).

Developer Response We have mitigated this by allowing individual users to manage the whitelisted sets of provers for the functions which they own, if they prefer a custom prover set to the ones provided by the guardians.

4.1.27 V-SCT-VUL-027: Contracts: All _callbackAddresses permitted

Severity	Warning	**Commit**	abd4356
Type	Data Validation	**Status**	Fixed
File(s)	`contracts/src/SuccinctGateway.sol`		
Location(s)	fulfillCall(), fulfillCallback()		
Confirmed Fix At	`https://github.com/succinctlabs/succinctx/pull/319`		

The `fulfillCall()` and `fulfillCallback()` functions are used by provers to complete submitted requests once verification has passed.

```
// Execute the callback.
isCallback = true;
(bool status,) =
    _callbackAddress.call(abi.encodeWithSelector(_callbackSelector, _output, _context
    ));
isCallback = false;
```

Snippet 4.28: Snippet from `fulfillCallback()` invoking the user-requested `_callbackAddress`.

While `requestCallback()` endpoint sets the `_callbackAddress` to `msg.sender`, a `requestCall` may request a call to ***any*** address.

Further, for many reasons, a `fulfillCallback()` may be invoked on an unexpected address. For example

1. An error in off-chain prover code may lead to calling the wrong address.
2. A malicious prover (see V-SCT-VUL-026) may invoke arbitrary addresses/functions.
3. A user may set the `_entryAddress` of a `requestCall()` to the address of the `SuccinctGateway` itself, causing the `SuccinctGateway` contract itself to call `requestCallback()`.

Impact Depending on future features added to the contract, its future holdings in other on-chain assets, or assumptions other contracts make about the `SuccinctGateway`'s behavior, unexpected behavior may occur.

Note that the impact is limited by the fact that these callbacks cannot send ether, and under normal operation conditions no other digital assets are owned by the contract.

Recommendation Allow whitelisting or blacklisting of callback addresses.

Developer Response Whitelist of allowed addresses to call would likely be too cumbersome for us to maintain, and it is not clear that a blacklist would help the situation much, as it would normally only be updated after an attack.

We have instead taken the approach of limiting the damage of what calling a malicious address could do (e.g. preventing re-entrancy). With this in mind, could you go over the worst-case scenario in more detail? What could a malicious address that gets called do to negatively impact the protocol and/or cause loss of funds?

Veridise Response We have not identified any current path by which the callbacks may be exploited.

Our primary concern was that, if new features to the contract were added without proper protection, users could cause the contract to call back into itself to abuse those new features. Since the contracts are no longer upgradeable, this is no longer a concern.

As the lack of upgradeability addresses our primary concern, we have included the corresponding PR as the resolution for this issue.

4.1.28 V-SCT-VUL-028: Replace GoldilocksHashOut size with named constant

Severity	Info	**Commit**	89b5a01
Type	Maintainability	**Status**	Fixed
File(s)	`poseidon/goldilocks.go`, `challenger/challenger.go`		
Location(s)	type GoldilocksHashOut, GetHash()		
Confirmed Fix At	7c7a01a		

The definition of `GoldilocksHashOut` does not use a named constant value to define the length of the underlying variable array.

Impact This leads to minor maintainability issues, such as having to propagate a hard-coded constant to other locations, such as `GetHash()` in `challenger/challenger.go`:

```
1 func (c *Chip) GetHash() poseidon.GoldilocksHashOut {
2         return [4]gl.Variable{c.GetChallenge(), c.GetChallenge(), c.GetChallenge(), c
      .GetChallenge()}
3 }
```

Recommendation Use a named constant value to refer to the length of the `GoldilocksHashOut` type.

Developer Response We applied the recommended fix.

4.1.29 V-SCT-VUL-029: Type error in bn254's HashOrNoop()

Severity	Info	**Commit**	89b5a01
Type	Maintainability	**Status**	Fixed
File(s)	`poseidon/bn254.go`		
Location(s)	HashOrNoop()		
Confirmed Fix At	40d71e9		

In `HashOrNoop`, the `inputElement` argument to `api.MulAcc` has an incorrect type. `api.MulAcc` expects a `frontend.Variable`, but `inputElement` has type `goldilocks.Variable`.

```
func (c *BN254Chip) HashOrNoop(input []gl.Variable) BN254HashOut {
    if len(input) <= 3 {
        returnVal := frontend.Variable(0)

        alpha := new(big.Int).SetInt64(1 << 32)
        for i, inputElement := range input {
            returnVal = c.api.MulAcc(returnVal, inputElement, alpha.Exp(alpha, big.
    NewInt(int64(i)), nil))
        }

        return BN254HashOut(returnVal)
    } else {
        return c.HashNoPad(input)
    }
}
```

Snippet 4.29: Snippet from `poseidon/bn254.go`

Impact This results in a circuit compilation error whenever `input` has length less than or equal to 3.

```
parsing circuit error="
goldilocks.Variable to big.Int not supported
frontend.parseCircuit.func2
    compile.go:118
utils.FromInterface
    convert.go:86
...
```

Recommendation Modify `inputElement` to `inputElement.Limb`.

Developer Response We applied the recommended fix.

4.1.30 V-SCT-VUL-030: Non-std range-check used

Severity	Info	**Commit**	89b5a01
Type	Maintainability	**Status**	Fixed
File(s)	goldilocks/base.go		
Location(s)	type Chip		
Confirmed Fix At	https://github.com/succinctlabs/gnark-plonky2-verifier/pull/47		

The goldilocks base `Chip` uses `frontend.Rangechecker` for range checks.

```
// The chip used for Goldilocks field operations.
type Chip struct {
    api          frontend.API
    rangeChecker frontend.Rangechecker
}
```

Snippet 4.30: Type declaration of goldilocks `Chip`.

The Gnark documentation for `frontend.Rangechecker` recommends:

> Users should instead use github.com/consensys/gnark/std/rangecheck package which automatically chooses most optimal method for range checking the variables.

Impact Certain compilers may not support this implementation.

Recommendation Use `std.rangecheck` instead of `frontend.Rangechecker`.

Developer Response We applied the recommended fix.

Updated Veridise Response Due to issue `https://github.com/advisories/GHSA-rjjm-x32p-m3f7` (see also `https://github.com/Consensys/gnark/issues/897` and https://github.com/Consensys/gnark/com we additionally recommend that the developers either wait for gnark version 0.9.2 or manually verify that this will not be an issue by adding additional checks at circuit compilation time.

Updated Developer Response We have added additional checks to ensure that the comparison performed is precise, even in the presence of the referenced issue.

4.1.31 V-SCT-VUL-031: Various out-of-date comments and documentation

Severity	Info	**Commit**	89b5a01
Type	Maintainability	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	f256ca6		

The following locations have out-of-date comments and documentation:

- `fri/fri.go`: Comment in `assertLeadingZeros` references non-existent variables from the reference implementation (`leading_zeros`; should instead reference `ProofOfWorkBits`).
- `go.mod`: Says that the required `go` version is 1.19, but the README says to use `go` version 1.20.1+.
- `goldilocks/base.go`: the comments of functions `Add`, `AddNoReduce`, `Sub`, `SubNoReduce`, `Mul`, `MulNoReduce`, `MulAdd`, `MulAddNoReduce` are out of date, and some are incorrect (argument names mismatched, incorrect arguments used, etc.).
- `goldilocks/quadratic_extensions.go`: the documentation claims that the function `SubMulExtension` constrains `a, b, c` such that `res = a*b - c`. However, the function introduces constraints for `res = (a - b)*c`.
- `goldilocks/quadratic_extensions.go`: the documentation claims that the function `Lookup2` is similar to gnark's `Select2` and that it operates on a variable. However, the function in gnark is actually also called `Lookup2`, and it operates on two variables.
- `plonk/gates/reducing_extension_gate.go`: the comment in `deserializeReducingExtensionGate` is copied from `ReducingGate`.
- `plonk/gates/random_access_gate.go`: lines 153-154, the comment claims that the added constrained are `bx - (by - y)` but the implementation adds `by - (bx - x)`.

Impact This can introduce bugs in future versions of the codebase when new developers mistakenly use the function for what the documentation is claiming.

Recommendation Update the documentation to reflect current the implementation.

Developer Response We applied the recommended fix.

4.1.32 V-SCT-VUL-032: Typos in code

Severity	Info	**Commit**	89b5a01
Type	Maintainability	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	de0ff4f		

The codebase contains a few minor typos, listed below.

- `zetaNextBath` in fri/fri.go.
- `If the bit is on` (should be "one") in fri/fri.go.
- `panic("len(evals) ! arity")` (should be "!=") in fri/fri.go.
- `aritheticExtensionGateRegex` in plonk/gates/arithmetic_extension_gate.go.
- `aritheticGateRegex` in plonk/gates/arithmetic_extension.go.

Developer Response We applied the recommended fix.

4.1.33 V-SCT-VUL-033: Ignored gate parameters

Severity	Info	**Commit**	89b5a01
Type	Maintainability	**Status**	Fixed
File(s)	`plonk/gates/{exponentiation_gate.go, random_access_gate.go}`		
Location(s)	exponentiationGateRegex, randomAccessGateRegex		
Confirmed Fix At	06f91e4		

Both the `ExponentiationGate` and `RandomAccessGate` capture a `base` parameter (corresponding to constant `D`) while deserializing the gate parameters. This parameter is then unused.

```
1 var exponentiationGateRegex = regexp.MustCompile("ExponentiationGate { num_power_bits
    : (?P<numPowerBits>[0-9]+), _phantom: PhantomData<plonky2_field::goldilocks_field
    ::GoldilocksField> }<D=(?P<base>[0-9]+)>")
```

Snippet 4.31: Example regex from `exponentiation_gate.go`.

Impact These ignored parameters may become an issue down the road if they no longer match the assumptions in the codebase.

Recommendation Either check that `parameters["base"]` matches the current assumptions of the codebase (i.e., `parameters["base"] == gl.D`), or add documentation explaining why checking `parameters["base"]` is unnecessary.

Developer Response We applied the recommended fix.

4.1.34 V-SCT-VUL-034: Hard-coded constants in code

Severity	Info	**Commit**	89b5a01
Type	Maintainability	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	e3eff27		

Certain parts of the code use hard-coded values rather than relying on global constants.

- `plonk/gates/poseidon_mds_gate.go`
  - In `EvalUnfiltered()`, a quadratic extension variable is constrained to be zero.

```
diff := glApi.SubExtensionAlgebra(output, computed_outputs[i])
constraints = append(constraints, diff[0], diff[1])
```

Snippet 4.32: Snippet from `EvalUnfiltered()`

If the extension degree is ever changed to be larger than 2, this will only constrain the first two components. As implemented elsewhere in the code, the preferred way to constrain a `QuadraticExtensionVariable` to be zero is to use a loop with bound `gl.D`.

- `plonk/gates/public_input_gate.go`
  - In `EvalUnfiltered()`, a loop uses a hardcoded constant for iterating over wires and hash values rather than iterating over the total number of wires (i.e., `len(wires)`). This could be problematic if the gate is ever modified to take a different number of input wires.

```
for i := 0; i < 4; i++ {
    wire := wires[i]
    hash_part := hash_parts[i]
    ...
}
```

Snippet 4.33: Snippet from `EvalUnfiltered()`

Impact If these constants change in the future, these implementations will become incorrect.

Recommendation Use the applicable constant.

Developer Response We applied the recommended fix.

4.1.35 V-SCT-VUL-035: No assertion that gnark is using BN254

Severity	Info	**Commit**	89b5a01
Type	Maintainability	**Status**	Fixed
File(s)	`poseidon/bn254.go`		
Location(s)	NewBN254Chip()		
Confirmed Fix At	4951161		

The BN254 Poseidon chip computes the Poseidon hash function over the BN254 scalar field. The implementation assumes the underlying field is the scalar field for BN254.

However, gnark supports several other fields, any of which may be used when compiling the constraints.

Impact If the incorrect field is used, the BN254 chip will no longer be computing a Poseidon hash over the BN254 field.

Recommendation Consider adding a check to `NewBN254Chip()` to ensure the `api.Compiler().Field()` is as expected.

Developer Response We applied the recommended fix.

4.1.36 V-SCT-VUL-036: Inaccurate bounds-check on numConsts

Severity	Info	**Commit**	89b5a01
Type	Data Validation	**Status**	Fixed
File(s)	`plonk/gates/constant_gate.go`		
Location(s)	ConstInput(), WireOutput()		
Confirmed Fix At	9e6d08b		

In both the `ConstInput` and `WireOutput` function, the input argument is checked to ensure it is not greater than the number of constants.

```
1 func (g *ConstantGate) ConstInput(i uint64) uint64 {
2       if i > g.numConsts {
3               panic("Invalid constant index")
4       }
5       return i
6 }
```

Snippet 4.34: Definition of `ConstInput()`.

However, since the return value is used as an array index into an array of size `g.numConsts`, `i` should be checked to ensure it is not greater than *or equal* to `g.numConsts` (i.e., `if i >= g.numConsts`).

Impact This could lead to an array out of bounds error if `i == g.numConsts`.

Recommendation Update the check on `i` to be `i >= g.numConsts`.

Developer Response We applied the recommended fix.

4.1.37 V-SCT-VUL-037: Sub-optimal sub-expression in random access gate

Severity	Info	**Commit**	89b5a01
Type	Constraint Optimization	**Status**	Fixed
File(s)	`plonk/gates/random_access_gate.go`		
Location(s)	EvalUnfiltered()		
Confirmed Fix At	8567f33		

The `EvalUnfiltered` function of the random access gate creates sub-optimal expressions while building the constraints. Specifically, for every two consecutive elements `(x, y)` in slice `listItems`, `EvalUnfiltered` creates the expression `by - (bx - x)`. However, this can be simplified to `x + b(y - x)`, which has one fewer operation.

```
1 mul1 := glApi.MulExtension(b, x)
2 sub1 := glApi.SubExtension(mul1, x)
3
4 mul2 := glApi.MulExtension(b, y)
5 sub2 := glApi.SubExtension(mul2, sub1)
6
7 listItemsTmp = append(listItemsTmp, sub2)
```

Snippet 4.35: Snippet from `EvalUnfiltered`

Impact Since every sub-expression will add several constraints, this logic will yield a sub-optimal overall circuit.

Recommendation We recommend simplifying the sub-expressions to `x + b(y - x)`.

Developer Response We applied the recommended fix.

4.1.38 V-SCT-VUL-038: Contracts: Code Recommendations

Severity	Info	**Commit**	abd4356
Type	Maintainability	**Status**	Fixed
File(s)	`contracts/src/FunctionRegistry.sol`		
Location(s)	FunctionRegistry		
Confirmed Fix At	`https://github.com/succinctlabs/succinctx/pull/315;https://github.com/succinctlabs/succinctx/pull/314`		

1. `FunctionRegistry` implements the `IFunctionRegistry` interface, but does not mark the functions as `override`.
2. `FunctionRegistry` contains several functions with similar functionality. For instance, `registerFunction()` registers a function, while `deployAndRegisterFunction()` first deploys a contract, and then registers it. As can be seen in the below snippet, the functions are nearly identical.

```
functionId = getFunctionId(_owner, _salt);
if (address(verifiers[functionId]) != address(0)) {
    revert FunctionAlreadyRegistered(functionId); // should call update instead
}
if (_verifier == address(0)) {
    revert VerifierCannotBeZero();
}
verifierOwners[functionId] = _owner;
verifiers[functionId] = _verifier;

emit FunctionRegistered(functionId, _verifier, _salt, _owner);
```

Snippet 4.36: Definition of `registerFunction()`

```
functionId = getFunctionId(_owner, _salt);
if (address(verifiers[functionId]) != address(0)) {
    revert FunctionAlreadyRegistered(functionId); // should call update instead
}

verifierOwners[functionId] = _owner;
verifier = _deploy(_bytecode, functionId);
verifiers[functionId] = verifier;

emit FunctionRegistered(functionId, verifier, _salt, _owner);
```

Snippet 4.37: Definition of `deployAndRegisterFunction()`

There is similar duplication between `updateFunction()` and `deployAndUpdateFunction()`.

Impact

1. If a function is ever deleted from the interface, the corresponding change may be forgotten in the `FunctionRegistry` contract.
2. Changes to the code may be made in only one function, leading to errors or inconsistencies down the road.

Recommendation Label the functions as overridden and remove the code duplication.

Developer Response

1. Functions marked as `override` in `SuccinctGateway` and `FunctionRegistry`.
2. Common functionality moved into internal `_register()` and `_update()` functions.

4.1.39 V-SCT-VUL-039: Contracts: Possible Wasted Gas

Severity	Info	**Commit**	abd4356
Type	Gas Optimization	**Status**	Fixed
File(s)	`contracts/src/FunctionRegistry.sol`, `contracts/src/SuccinctGateway.sol`		
Location(s)	See Issue Description		
Confirmed Fix At	`https://github.com/succinctlabs/succinctx/pull/313`, `https://github.com/succinctlabs/succinctx/pull/315`		

When a user submits a transaction to the Ethereum blockchain for processing, they specify the maximum gas fee they are willing to pay. The complexity of the computation in the transaction determines the amount of gas used to process the transaction.

1. The Solidity language provides the `delete` keyword to reset variables to the default value for the data type. Using the `delete` keyword also refunds a small amount of the gas fee spent on the transaction. The `SuccinctGateway.removeProver()` function could use `delete allowedProvers[_prover]` instead of `allowedProvers[_prover] = false` since the default value for the bool data type is `false`.
2. To save on gas fees in the case where a transaction reverts, it is best for the conditions that could cause a revert to be checked as soon as possible. The `FunctionRegistry` contract has two functions where the revert-case gas cost could be optimized by checking the simpler `_verifier == address(0)` condition before computing the `functionId`.

```
functionId = getFunctionId(_owner, _name);
if (address(verifiers[functionId]) != address(0)) {
    revert FunctionAlreadyRegistered(functionId); // should call update instead
}
if (_verifier == address(0)) {
    revert VerifierCannotBeZero();
}
```

Snippet 4.38: Snippet from `FunctionRegistry.registerFunction()`

```
functionId = getFunctionId(msg.sender, _name);
if (msg.sender != verifierOwners[functionId]) {
    revert NotFunctionOwner(msg.sender, verifierOwners[functionId]);
}
if (_verifier == address(0)) {
    revert VerifierCannotBeZero();
}
```

Snippet 4.39: Snippet from `FunctionRegistry.updateFunction()`

3. The `addProver()` and `removeProver()` functions in the `SuccinctGateway` contract are used to manage the addresses in the `allowedProvers` map by adding/removing the given address. The presence or absence of `address(0)` in the `allowedProvers` map has no meaningful effect. Thus, both functions could allow users to avoid spurious executions before submitting the transaction by reverting when `_prover == address(0)`.

```
function addProver(address _prover) external onlyGuardian {
    allowedProvers[_prover] = true;
    emit ProverUpdated(_prover, true);
}
```

Snippet 4.40: Definition of `SuccinctGateway.addProver()`. `removeProver()` is similar.

```
modifier onlyProver() {
    if (!allowedProvers[msg.sender]) {
        revert OnlyProver(msg.sender);
    }
    _;
}
```

Snippet 4.41: Definition of `SuccinctGateway.onlyProver()` modifier

Impact Higher than necessary gas fees may be required in several functions.

Recommendation

1. In `SuccinctGateway.removeProver()`, replace `allowedProvers[_prover] = false` with `delete allowedProvers[_prover]`.
2. In `FunctionRegistry.registerFunction()` and `FunctionRegistry.updateFunction()`, move the `_verifier == address(0)` check to the start of the function.
3. In `SuccinctGateway.addProver()` and `SuccinctGateway.removeProver()` add a condition to revert if `_prover == address(0)`.

Developer Response

1. We applied recommendation.
2. We applied recommendation.
3. The `addProver()` and `removeProver()` functions were removed from `SuccinctGateway` in https://github.com/succinctlabs/succinctx/pull/310 and replaced with `addCustomProver()`, `addDefaultProver()`, `removeCustomProver()`, and `removeDefaultProver()`. However, none of these functions require the condition `_prover != address(0)`.

5 Fuzz Testing

5.1 Methodology

Veridise auditors set out to fuzz test the witness generation in gnark-plonky2-verifier to increase confidence in its implementation and gain greater assurance in the correctness of constraints which have been determined (by manual review) to match the witness generator. These fuzz tests focused on functional correctness i.e, whether the implementation deviates from the intended behavior.

The auditors primarily performed differential fuzzing–running an alternative implementation on the same inputs and then comparing the results. Some of these alternative implementations were hand-crafted to match the desired behavior, and others were taken from popular repositories such as `https://github.com/iden3/circomlib` and `https://github.com/iden3/go-iden3-crypto`. See section (5.3) for more details.

5.2 Properties Fuzzed

Table 5.1 describes the invariants we fuzz-tested. The second column states which component (e.g., correct emulation of addition in the Goldilocks field) the invariant is associated with. The third shows the total amount of compute time spent fuzzing this property. The last column notes whether we found a bug when fuzzing the invariant (✗ indicates no bug was found and ✓ means fuzzing this invariant revealed a bug).

The Veridise auditors devoted a total of 303 compute-hours to fuzzing this protocol, identifying a total of 0 bugs.

Table 5.1: Invariants Fuzzed.

Specification	Invariant	Minutes Fuzzed	Bugs Found
V-SCT-SPEC-001	Spec: base — Add	120	✗
V-SCT-SPEC-002	Spec: base — Exp	120	✗
V-SCT-SPEC-003	Spec: base — Inverse	120	✗
V-SCT-SPEC-004	Spec: base — Mul	120	✗
V-SCT-SPEC-005	Spec: base — MulAdd	120	✗
V-SCT-SPEC-006	Spec: base — RangeCheck	60	✗
V-SCT-SPEC-007	Spec: base — Reduce	120	✗
V-SCT-SPEC-008	Spec: poseidon — bn254	8000	✗
V-SCT-SPEC-009	Spec: poseidon — goldilocks	8000	✗
V-SCT-SPEC-010	Spec: quadratic_extension — AddExtension	120	✗
V-SCT-SPEC-011	Spec: quadratic_extension — ExpExtension	120	✗
V-SCT-SPEC-012	Spec: quadratic_extension — InnerProductExten	120	✗
V-SCT-SPEC-013	Spec: quadratic_extension — InverseExtension	120	✗
V-SCT-SPEC-014	Spec: quadratic_extension — IsZero	120	✗
V-SCT-SPEC-015	Spec: quadratic_extension — Lookup	120	✗
V-SCT-SPEC-016	Spec: quadratic_extension — Lookup2	120	✗

V-SCT-SPEC-017	Spec: quadratic_extension — MulAddExtension	120	✗
V-SCT-SPEC-018	Spec: quadratic_extension — MulExtension	120	✗
V-SCT-SPEC-019	Spec: quadratic_extension — ScalarMulExtensio	120	✗
V-SCT-SPEC-020	Spec: quadratic_extension — SubExtension	120	✗
V-SCT-SPEC-021	Spec: quadratic_extension — SubMulExtension	120	✗

5.3 Detailed Description of Fuzzed Specifications

In the rest of this section, M denotes the Goldilocks prime.

5.3.1 V-SCT-SPEC-001: Spec: base — Add

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `Add()` function in `goldilocks/base.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results.

Formal Specification We checked the output of `Add` satisfies the below formula.

$$\text{Add}(x, y) = (x + y) \mod M.$$

5.3.2 V-SCT-SPEC-002: Spec: base — Exp

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `Exp()` function in `goldilocks/base.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results. We in particular use a naive exponentiation by squaring to implement the Go function.

Formal Specification We checked the output of `Exp` satisfies the below formula.

$$\text{Exp}(x, n) = x^n \mod M.$$

5.3.3 V-SCT-SPEC-003: Spec: base — Inverse

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `Inverse()` function in `goldilocks/base.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results. We in particular use a naive exponentiation by squaring to implement the Go function.

Formal Specification We checked the output of `Inverse` satisfies the below formula.

Inverse(x) = y such that $xy \mod M = 1$.

But to actually compute y, we use Fermat's little theorem:

$$x^M = x \pmod M$$
$$x^{M-2} = x^{-1} \pmod M$$

where we use exponentiation by squaring for efficiency.

5.3.4 V-SCT-SPEC-004: Spec: base — Mul

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `Mul()` function in `goldilocks/base.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results.

Formal Specification We checked the output of `Mul` satisfies the below formula.

$$\text{Mul}(x, y) = (x \cdot y) \mod M.$$

5.3.5 V-SCT-SPEC-005: Spec: base — MulAdd

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `MulAdd()` function in `goldilocks/base.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results.

Formal Specification We checked the output of `MulAdd` satisfies the below formula.

$$\text{MulAdd}(x, y, z) = ((x \cdot y) + z) \mod M.$$

5.3.6 V-SCT-SPEC-006: Spec: base — RangeCheck

Minutes Fuzzed	60	Bugs Found	0

Natural Language We fuzzed the witness generation for the `RangeCheck()` function in `goldilocks/base.go`. We tested for functional correctness by checking if the predicate is satisfied on various inputs, using a naive comparison as a ground truth.

Formal Specification We checked the `RangeCheck` is equivalent the the following predicate:

RangeCheck(x) iff $x < M$.

5.3.7 V-SCT-SPEC-007: Spec: base — Reduce

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `Reduce()` function in `goldilocks/base.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results.

Formal Specification We checked the output of `Reduce` satisfies the below formula.

$$\text{Reduce}(x) = x \mod M$$

5.3.8 V-SCT-SPEC-008: Spec: poseidon — bn254

Minutes Fuzzed	8000	Bugs Found	0

Natural Language We fuzzed the witness generation for the `Poseidon()` function in `poseidon/bn254.go`. We tested for functional correctness by computing the expected value using Circomlib's implementation at

`https://github.com/iden3/circomlib/blob/cff5ab6288b55ef23602221694a6a38a0239dcc0/circuits/poseidon.circom`

and comparing the two results. The Circomlib's template is instantiated with `PoseidonEx(3, 4)`.

5.3.9 V-SCT-SPEC-009: Spec: poseidon — goldilocks

Minutes Fuzzed	8000	Bugs Found	0

Natural Language We fuzzed the witness generation for the `Poseidon()` function in `poseidon/goldilocks.go`. We tested for functional correctness by computing the expected value using Go's implementation at

`https://github.com/iden3/go-iden3-crypto/blob/3fb23d780c02f41857d62ffa04c3a12912b14761/goldenposeidon/poseidon.go#L101`

and comparing the two results. The Go function is manually modified to output 12 elements instead of 4.

5.3.10 V-SCT-SPEC-010: Spec: quadratic_extension — AddExtension

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `AddExtension()` function in `goldilocks/quadratic_extension.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results.

Formal Specification We checked the output of `AddExtension` satisfies the below formula.

$$\text{AddExtension}(a + b\sqrt{7}, c + d\sqrt{7}) = ((a + c) \mod M) + ((b + d) \mod M)\sqrt{7}.$$

5.3.11 V-SCT-SPEC-011: Spec: quadratic_extension — ExpExtension

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `ExpExtension()` function in `goldilocks/quadratic_extension.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results. In particular, we use a naive exponentiation based on repeated squaring to implement the Go function.

Formal Specification We checked the output of `ExpExtension` satisfies the below formula.

$$\text{ExpExtension}(a + b\sqrt{7}, n) = (a + b\sqrt{7})^n$$

5.3.12 V-SCT-SPEC-012: Spec: quadratic_extension — InnerProductExtension

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `InnerProductExtension()` function in `goldilocks/quadratic_extension.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results.

Formal Specification We checked the output of `InnerProductExtension` satisfies the below formula.

InnerProductExtension = f

$$
\begin{aligned}
&f(c, a + b\sqrt{7}, [\langle x_1 + y_1\sqrt{7}, u_1 + v_1\sqrt{7}\rangle, \ldots]) = \\
&\quad ((a + cx_1u_1 + 7cy_1v_1 + cx_2u_2 + 7cy_2v_2 + \ldots) \mod M) + \\
&\quad ((b + cx_1v_1 + cy_1u_1 + cx_2v_2 + cy_2u_2 + \ldots) \mod M)\sqrt{7}
\end{aligned}
$$

5.3.13 V-SCT-SPEC-013: Spec: quadratic_extension — InverseExtension

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `InverseExtension()` function in `goldilocks/quadratic_extension.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results.

Formal Specification We checked the output of `InverseExtension` satisfies the below formula.

InverseExtension$(a+b\sqrt{7}) = c+d\sqrt{7}$ such that $ac+7cd = 1 \pmod M \wedge ad+bc = 0 \pmod M$

But to actually compute $c + d\sqrt{7}$, we follow:

$$\frac{1}{a+b\sqrt{7}} = \frac{1}{a+b\sqrt{7}} \cdot \frac{a-b\sqrt{7}}{a-b\sqrt{7}} = \frac{a-b\sqrt{7}}{a^2-7b^2}$$

The inverse of $a^2 - 7b^2$ under M can be found by using Fermat's little theorem:

$$x^M = x \pmod M$$
$$x^{M-2} = x^{-1} \pmod M$$

where we use exponentiation by squaring for efficiency.

Note that $a = b = 0$ is the only case where the inverse is not defined. $a^2 = 7b^2 \pmod M$ doesn't have any other solutions.

5.3.14 V-SCT-SPEC-014: Spec: quadratic_extension — IsZero

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `IsZero()` function in `goldilocks/quadratic_extension.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results.

Formal Specification We checked the output of `IsZero` satisfies the below formula.

IsZero = f where

$$f(a + b\sqrt{7}) = \begin{cases} 1 & \text{if } a = 0 \wedge b = 0 \\ 0 & \text{otherwise} \end{cases}$$

5.3.15 V-SCT-SPEC-015: Spec: quadratic_extension — Lookup

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `Lookup()` function in `goldilocks/quadratic_extension.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results.

Formal Specification We checked the output of `Lookup` satisfies the below formula.

Lookup = f where

$$f(b,u,v) = \begin{cases} u & \text{if } b = 0 \\ v & \text{if } b = 1 \end{cases}$$

5.3.16 V-SCT-SPEC-016: Spec: quadratic_extension — Lookup2

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `Lookup2()` function in `goldilocks/quadratic_extension.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results.

Formal Specification We checked the output of `Lookup2` satisfies the below formula.

Lookup2 = f where

$$f(b_1, b_2, u, v, x, y) = \begin{cases} u & \text{if } b_1 = 0 \wedge b_2 = 0 \\ v & \text{if } b = 1 \wedge b_2 = 0 \\ x & \text{if } b = 0 \wedge b_2 = 1 \\ y & \text{if } b = 1 \wedge b_2 = 1 \end{cases}$$

5.3.17 V-SCT-SPEC-017: Spec: quadratic_extension — MulAddExtension

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `MulAddExtension()` function in `goldilocks/quadratic_extension.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results.

Formal Specification We checked the output of `MulAddExtension` satisfies the below formula.

$\text{MulAddExtension}(a + b\sqrt{7}, c + d\sqrt{7}, e + f\sqrt{7}) = ((ac + 7bd + e) \mod M) + ((ad + bc + f) \mod M)\sqrt{7}$.

5.3.18 V-SCT-SPEC-018: Spec: quadratic_extension — MulExtension

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `MulExtension()` function in `goldilocks/quadratic_extension.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results.

Formal Specification We checked the output of `MulExtension` satisfies the below formula.

$$\text{MulExtension}(a + b\sqrt{7}, c + d\sqrt{7}) = ((ac + 7bd) \mod M) + ((ad + bc) \mod M)\sqrt{7}.$$

5.3.19 V-SCT-SPEC-019: Spec: quadratic_extension — ScalarMulExtension

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `ScalarMulExtension()` function in `goldilocks/quadratic_extension.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results.

Formal Specification We checked the output of `ScalarMulExtension` satisfies the below formula.

$$\text{ScalarMulExtension}(a + b\sqrt{7}, c) = (ac \mod M) + (bc \mod M)\sqrt{7}.$$

5.3.20 V-SCT-SPEC-020: Spec: quadratic_extension — SubExtension

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `SubExtension()` function in `goldilocks/quadratic_extension.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results.

Formal Specification We checked the output of `SubExtension` satisfies the below formula.

$$\text{SubExtension}(a + b\sqrt{7}, c + d\sqrt{7}) = ((a - c + M) \mod M) + ((b - d + M) \mod M)\sqrt{7}.$$

5.3.21 V-SCT-SPEC-021: Spec: quadratic_extension — SubMulExtension

Minutes Fuzzed	120	Bugs Found	0

Natural Language We fuzzed the witness generation for the `SubMulExtension()` function in `goldilocks/quadratic_extension.go`. We tested for functional correctness by computing the expected value in a separate Go function and comparing the two results.

Formal Specification We checked the output of `SubMulExtension` satisfies the below formula.

$$\text{SubMulExtension}(a+b\sqrt{7}, c+d\sqrt{7}, e+f\sqrt{7}) = ((xe+7yf) \mod M) + ((xf+ye) \mod M)\sqrt{7}$$

where $x = ((a-c+M) \mod M)$ and $y = (b-d+M) \mod M$.

6 Formal Verification

6.1 Formal Verification Procedure

Our team verified several properties of the code. See Table 6.1 for a complete list.

We formally verified the code using Veridise's tool Picus. Picus is an open-source library that can be used to prove that zero-knowledge circuits are properly constrained* by formally verifying they have exactly one solution or find a counterexample if the circuits are not properly constrained. It implements this by a novel interaction between static analysis and SMT solvers.

Veridise auditors ran Picus on several key functions. The effort focused on the Goldilocks field emulation as most of the codebase depends on that module, and the most severe issues were identified in that region of the code (see V-SCT-VUL-001, V-SCT-VUL-002, and V-SCT-VUL-003). They also ran the tool on the quadratic extension, quadratic extension algebra, and Poseidon hash (both BN254 and Goldilocks).

To improve scalability, we modeled some constraints with a formula that can be directly reasoned with by SMT solvers. For example, we modeled several range checking constraints by replacing them with a simple inequality assertion.

Furthermore, identified under-constrained circuits (Subsections 4.1.1 to 4.1.3) could cause other circuits that include the under-constrained circuits as sub-circuits to become under-constrained as well, resulting in redundant reports. We made a fix to these under-constrained circuits first before proceeding to verify other circuits.

6.2 Properties Verified

A complete list of the properties verified is shown in Table 6.1. Each row displays a natural language description of the property proved, and its current status (i.e. verified, not verified).

Table 6.1: Formally verified properties.

ID	Property	Status
V-SCT-FSPEC-001	Determinism of `MulAdd` File: `goldilocks/base.go`	Verified
V-SCT-FSPEC-002	Determinism of `ReduceWithMaxBits` File: `goldilocks/base.go`	Unknown†
V-SCT-FSPEC-003	Determinism of `Exp` File: `goldilocks/base.go`	Verified
V-SCT-FSPEC-004	Determinism of `Inverse` File: `goldilocks/base.go`	Not verified‡
V-SCT-FSPEC-005	Determinism of `AddExtension` File: `goldilocks/quadratic_extension.go`	Verified

* In this context, *properly constrained* means not under-constrained.
† Discovered to be not verified manually – see Subsection 4.1.3
‡ See Subsection 4.1.1

V-SCT-FSPEC-006	Determinism of `SubExtension` File: `goldilocks/quadratic_extension.go`	Verified
V-SCT-FSPEC-007	Determinism of `MulExtension` File: `goldilocks/quadratic_extension.go`	Verified
V-SCT-FSPEC-008	Determinism of `MulAddExtension` File: `goldilocks/quadratic_extension.go`	Verified
V-SCT-FSPEC-009	Determinism of `SubMulExtension` File: `goldilocks/quadratic_extension.go`	Verified
V-SCT-FSPEC-010	Determinism of `ScalarMulExtension` File: `goldilocks/quadratic_extension.go`	Verified
V-SCT-FSPEC-011	Determinism of `InnerProductExtension` File: `goldilocks/quadratic_extension.go`	Verified
V-SCT-FSPEC-012	Determinism of `InverseExtension` File: `goldilocks/quadratic_extension.go`	Verified
V-SCT-FSPEC-013	Determinism of `DivExtension` File: `goldilocks/quadratic_extension.go`	Verified
V-SCT-FSPEC-014	Determinism of `ExpExtension` File: `goldilocks/quadratic_extension.go`	Verified
V-SCT-FSPEC-015	Determinism of `ReduceExtension` File: `goldilocks/quadratic_extension.go`	Verified
V-SCT-FSPEC-016	Determinism of `ReduceWithPowers` File: `goldilocks/quadratic_extension.go`	Verified
V-SCT-FSPEC-017	Determinism of `IsZero` File: `goldilocks/quadratic_extension.go`	Verified
V-SCT-FSPEC-018	Determinism of `Lookup` File: `goldilocks/quadratic_extension.go`	Verified
V-SCT-FSPEC-019	Determinism of `Lookup2` File: `goldilocks/quadratic_extension.go`	Verified
V-SCT-FSPEC-020	Determinism of `MulExtensionAlgebra` File: `goldilocks/quadratic_extension_algebra.go`	Verified
V-SCT-FSPEC-021	Determinism of `PartialInterpolateExtAlgebra` File: `goldilocks/quadratic_extension_algebra.go`	Verified
V-SCT-FSPEC-022	Determinism of `Poseidon` File: `poseidon/bn254.go`	Verified
V-SCT-FSPEC-023	Determinism of `HashNoPad` File: `poseidon/bn254.go`	Verified
V-SCT-FSPEC-024	Determinism of `HashOrNoOp` File: `poseidon/bn254.go`	Crashed[§]
V-SCT-FSPEC-025	Determinism of `TwoToOne` File: `poseidon/bn254.go`	Verified
V-SCT-FSPEC-026	Determinism of `ToVec` File: `poseidon/bn254.go`	Unknown
V-SCT-FSPEC-027	Determinism of `Poseidon` File: `poseidon/goldilocks.go`	Unknown

[§] See Subsection 4.1.29

Glossary

Fast Reed-Solomon IOPPs A transparent, succinct argument to prove [1, 2] a committed function is near a polynomial of low-degree. This is frequently used to produce a polynomial commitment scheme. See `https://vitalik.ca/general/2017/11/22/starks_part_2.html` for more details . 7, 87

FRI Fast Reed-Solomon IOPPs. 87

gnark A framework for zero-knowledge circuits. See `https://docs.gnark.consensys.net` to learn more . 1

Go An open-source language supported by Google popular in the blockchain ecosystem. See `https://go.dev` to learn more. 1, 8

Goldilocks Field The 64-bit field used in plonky2. 2, 7

PLONK An arithmetization strategy for zero-knowledge circuits developed in [3]. See [4] or `https://vitalik.ca/general/2019/09/22/plonk.html` for more details. 7, 87

PLONKish A catch-all term for arithmetization strategies for zero-knowledge circuits based off of PLONK, but slightly generalized (typically to handle more gates or a different polynomial commitment). For more details, see `https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-turbo_plonk.pdf`. 87

plonky2 A PLONKish arithmetization developed by Polygon Zero with various optimizations and FRI as the polynomial commitment scheme. See `https://github.com/0xPolygonZero/plonky2/` for more details . 1, 87

Satisfiability Modulo Theories The problem of determining whether a certain mathematical statement has any solutions. SMT solvers attempt to do this automatically. See `https://en.wikipedia.org/wiki/Satisfiability_modulo_theories` to learn more . 87

smart contract A self-executing contract with the terms directly written into code. Hosted on a blockchain, it automatically enforces and executes the terms of an agreement between buyer and seller. Smart contracts are transparent, tamper-proof, and eliminate the need for intermediaries, making transactions more efficient and secure.. 1, 87

SMT Satisfiability Modulo Theories. 85

Solidity The standard high-level language used to develop smart contracts on the Ethereum blockchain. See `https://docs.soliditylang.org/en/v0.8.19/` to learn more. 1

zero-knowledge circuit A cryptographic construct that allows a prover to demonstrate to a verifier that a certain statement is true, without revealing any specific information about the statement itself. See `https://en.wikipedia.org/wiki/Zero-knowledge_proof` for more . 7, 87

Bibliography

[1] Eli Ben-Sasson et al. 'Fast Reed-Solomon Interactive Oracle Proofs of Proximity'. In: *Electron. Colloquium Comput. Complex.* 2017 (cited on page 87).

[2] Eli Ben-Sasson et al. *DEEP-FRI: Sampling Outside the Box Improves Soundness*. Cryptology ePrint Archive, Paper 2019/336. https://eprint.iacr.org/2019/336. 2019. URL: https://eprint.iacr.org/2019/336 (cited on page 87).

[3] Ariel Gabizon, Zachary J. Williamson, and Oana-Madalina Ciobotaru. 'PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge'. In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 953 (cited on page 87).

[4] Justin Thaler. 'Proofs, Arguments, and Zero-Knowledge'. In: *Foundations and Trends® in Privacy and Security* 4.2–4 (2022), pp. 117–660. DOI: 10.1561/3300000030 (cited on page 87).

