Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Succinct Labs Telepathy

Veridise Inc.

March 11, 2023

► **Prepared For:**

Succinct Labs
https://succinct.xyz/

► **Prepared By:**

Kostas Ferles
Shankara Pailoor
Alp Bassa
Stefanos Chaliasos
Hanzhi Liu

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

► **Version History:**

January 19, 2023 V1
March 11, 2023 V2

Contents

1 Executive Summary

From October 25 to December 10, Succinct Labs engaged Veridise to review the security of the circom circuits used in the implementation of Telepathy, the first decentralized and permissionless interoperability layer for Ethereum. The review covered all circuits implemented by Succinct Labs engineers as well as circuits from the circom-pairing library, which was heavily utilized within Telepathy. Veridise conducted the assessment over 25 person-weeks, with 5 engineers reviewing code over 5 weeks on commit `f61894e`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Code assessment. Succinct Labs' Telepathy project is a decentralized and permissionless interoperability layer for Ethereum. Specifically, Telepathy allows one blockchain to verify the validity of accepted blocks in another blockchain in a decentralized manner. At its current state, Telepathy is designed for blockchains whose consesus protocol is based on Proof of Stake (PoS). The main challenge in designing such a layer is verifying the validity of blocks accepted in a different chain. To do that for a PoS blockchain, one would need the ability to verify cryptographic signatures in an efficient way. To tackle this challenge, Succinct Labs has employed Zero-Knowledge circuits, which significantly reduce the amount of computation performed on chain. Finally, it is worth mentioning that Succinct Labs has already used Telepathy to implement a bridge between several chains.

Succinct Labs provided the source code for the Telepathy circuits for review. Additionally, we were also pointed to the version of the circom-pairing library used by Telepathy, since it was also in scope of this audit. Both projects were accompanied by adequate documentation and extensive test suites. The intended behavior for the circuits was further communicated on calls with the client.

Summary of issues detected. The audit uncovered 36 issues, 4 of which are assessed to be of critical severity by the Veridise auditors. Specifically, bug V-SUC-VUL-002 is a missing input validation for a crucial circuit in the circom-pairing library that is used to verify signatures. This omission would have exposed a huge attack surface that malicious actors could exploit in order to forge signatures. Another critical bug discovered by Veridise engineers allowed attackers to control the rotation of the sync committee, which would allow them again to forge signatures and validate invalid blocks. The Veridise auditors also identified several moderate-severity issues, including missing checks when calculating aggregate signatures (V-SUC-VUL-007), circuits with invalid instantiations (V-SUC-VUL-005, V-SUC-VUL-008, V-SUC-VUL-010, V-SUC-VUL-011, V-SUC-VUL-012), and many more (see Section 4).

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

liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Succinct Labs Telepathy	`f61894e`	Circom	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Oct. 25 - Dec. 10, 2022	Manual & Tools	5	25 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	4	4
High-Severity Issues	0	0
Medium-Severity Issues	4	4
Low-Severity Issues	21	21
Warning-Severity Issues	4	4
Informational-Severity Issues	3	3
TOTAL	36	36

Table 2.4: Category Breakdown.

Name	Number
Logic Error	7
Maintainability	19
Optimization	2
Data Validation	8

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of the circuit portion of Telepathy. In our audit, we sought to answer the following questions:

- Are all circuits properly constrained? In other words, are there cases where the verifier can accept more solutions than anticipated?
- Do all circuits always generate the expected witness?
- Can malicious users forge signatures of the sync committee?
- Can users tamper with Telepathy's protocol execution?

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 static analyzer Vanguard, which supports circom circuits. Currently, Vanguard is equipped with detectors designed to find instances of common circuit vulnerabilities, such as unconstrained input or output signals.
- *Fuzzing.* We also leveraged fuzz testing to determine if the circom-pairing circuits may deviate from their expected behavior. To do this, we used existing elliptic pairing libraries as oracles and then used our custom fuzzing framework to determine if a violation of the specification can be found. Our fuzzer automatically tested several core components of the pairing for several days. In total, our fuzzer ran for more than 6 millions iterations across all tested circuits.

Scope. This audit reviewed the circom circuits of Telepathy, including the circuits from the circom-pairing library. As such, Veridise auditors first inspected the provided tests to better understand the desired behavior of the provided circuits at a more granular level. They then began a multi-week manual audit of the code assisted by both static analyzers and automated testing.

In terms of the audit, the key components include the following:

- The Telepathy protocol circuits.
- Circuits in the circom-pairing library used by Telepathy.

Limitations. Due to the scope of our audit, the recommendations provided in this report are limited to the functional specification provided by the Succinct Labs developers. The overall security of the system can be compromised if any component outside the scope of the audit is vulnerable. For Telepathy, such components include, but are not limited to, the following:

- **Circuit deployment**: If the circuits are not deployed according to industry standards, i.e., following a secure trusted setup ceremony, the whole protocol can be at risk in case the information used in the creation of the common reference string (CRS) is leaked.
- **Smart Contracts**: Security can also be compromised if the smart contracts that verify proofs contain bugs. As the smart contracts of Telepathy is outside the scope of this audit, this report does not make any guarantees to that extend.

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:

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:

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., acknowleged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-SUC-VUL-001	ArrayXOR is under constrained	Critical	Fixed
V-SUC-VUL-002	No input validation in template CoreVerifyPubkeyG1	Critical	Fixed
V-SUC-VUL-003	Zero padding vulnerable to overflow	Critical	Fixed
V-SUC-VUL-004	Sync committee can be rotated with random public keys	Critical	Fixed
V-SUC-VUL-005	LongToShortNoEndCarry has incomplete instantiations	Medium	Fixed
V-SUC-VUL-006	ModSumFour is Incorrect	Medium	Fixed
V-SUC-VUL-007	G1AddMany assumes no intermediate sum equal to the point	Medium	Fixed
V-SUC-VUL-008	VerifySyncCommittee lacking 0 check on aggregation bits	Medium	Fixed
V-SUC-VUL-009	Templates have non-deterministic instantiations	Low	Fixed
V-SUC-VUL-010	Implicit assumption in BigMult	Low	Fixed
V-SUC-VUL-011	BigMod does not work for all instantiations	Low	Fixed
V-SUC-VUL-012	BigMod2 does not work for all instantiations	Low	Fixed
V-SUC-VUL-013	BigModInv does not work for all instantiations	Low	Fixed
V-SUC-VUL-014	Implicit Assumption in BigModInv	Low	Fixed
V-SUC-VUL-015	Constant 512 should be SYNC-COMMITTEE-SIZE instead	Low	Fixed
V-SUC-VUL-016	Fp2Invert does not work for all valid instantiations	Low	Fixed
V-SUC-VUL-017	SignedFp2Divide does not work for all valid instantiations	Low	Fixed
V-SUC-VUL-018	Constant 32 should be B-IN-BYTES in ExpandMessageXMD	Low	Fixed
V-SUC-VUL-019	PrimeReduce does not work for all valid instantiations	Low	Fixed
V-SUC-VUL-020	SSZPhase0SyncCommittee should use template parameters	Low	Fixed
V-SUC-VUL-021	SSZLayer assumes numBytes Greater than Or Equal 64	Low	Fixed
V-SUC-VUL-022	SSZRestoreMerkelRoot signal value is unused	Low	Fixed
V-SUC-VUL-023	EllipticCurveAddUnequal doesn't work for all instantiations	Low	Fixed
V-SUC-VUL-024	Fp12Invert doesn't work for all instantiations	Low	Fixed
V-SUC-VUL-025	SignedCheckCarryModToZero doesn't work for instantiations	Low	Fixed
V-SUC-VUL-026	long-add4 doesn't work for all invocations	Low	Fixed
V-SUC-VUL-027	long-add-unequal doesn't work for all invocations	Low	Fixed
V-SUC-VUL-028	SignedFpCarryModP doesn't work for all instantiations	Low	Fixed
V-SUC-VUL-029	Long-add-unequal assumptions should be asserted	Low	Fixed
V-SUC-VUL-030	Intermediate Signal add.out is under constrained	Warning	Won't Fix
V-SUC-VUL-031	HashToField is under constrained	Warning	Fixed
V-SUC-VUL-032	Make all depth constants the floorlog2 of index constants	Warning	Won't Fix
V-SUC-VUL-033	Parameter N is unused in template PoseidonG1Array	Warning	Won't Fix
V-SUC-VUL-034	Input aggregatePubkeyBigInt can be removed	Info	Fixed
V-SUC-VUL-035	Duplicate Code Snippet	Info	Won't Fix
V-SUC-VUL-036	Gratuitous constraints in SerializeLightClientStepInputs	Info	Won't Fix

4.1 Detailed Description of Bugs

4.1.1 V-SUC-VUL-001: ArrayXOR is under constrained

Severity	Critical	**Commit**	f61894ea121d9b1b885
Type	Logic Error	**Status**	Fixed
Files	circuits/circuits/hash_to_field.circom		
Functions	Template ArrayXOR		

Template `ArrayXOR` is not properly constrained due to the use of operator `<--` (see attached snippet).

```
for (var i = 0; i < n; i++) {
    // xors[i] = XOR();
    // xors[i].a <== a[i];
    // xors[i].b <== b[i];
    out[i] <-- a[i] ^ b[i];
}
```

Impact The ArrayXOR circuit is used to hash a message to a field element. By leaving this circuit underconstrained, an attacker can select nearly any field element as the hash of the message, allowing them to forge the sync committee's BLS signatures.

Recommendation Replace operator `<--` with equivalent operations that use operator `<==`.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.2 V-SUC-VUL-002: Template CoreVerifyPubkeyG1 does not perform input validation

Severity	Critical	**Commit**	f61894ea121d9b1b885
Type	Logic Error	**Status**	Fixed
Files	circuits/pairing/bls_signature.circom		
Functions	Template CoreVerifyPubkeyG1		

Template `CoreVerifyPubkeyG1` instantiates several `BigLessThan` templates whose output signal is never used.

```
    component lt[10];
    // check all len k input arrays are correctly formatted bigints < q (BigLessThan
    calls Num2Bits)
    for(var i=0; i<10; i++){
        lt[i] = BigLessThan(n, k);
        for(var idx=0; idx<k; idx++)
            lt[i].b[idx] <== q[idx];
    }
    for(var idx=0; idx<k; idx++){
        lt[0].a[idx] <== pubkey[0][idx];
        lt[1].a[idx] <== pubkey[1][idx];
        lt[2].a[idx] <== signature[0][0][idx];
        lt[3].a[idx] <== signature[0][1][idx];
        lt[4].a[idx] <== signature[1][0][idx];
        lt[5].a[idx] <== signature[1][1][idx];
        lt[6].a[idx] <== hash[0][0][idx];
        lt[7].a[idx] <== hash[0][1][idx];
        lt[8].a[idx] <== hash[1][0][idx];
        lt[9].a[idx] <== hash[1][1][idx];
    }
```

Impact The aim of this check is to ensure that all inputs satisfy the assumptions made by the core circom-pairing circuits. Skipping this check increases the attack surface of the code base significantly.

Recommendation Ensure that the output signal of every template in array `lt` are properly constrained.

Developer Response The developers acknowledged and fixed the issue here.

4.1.3 V-SUC-VUL-003: Zero Padding for Sha256 in ExpandMessageXMD is vulnerable to an overflow

Severity	Critical	Commit	f61894ea121d9b1b885
Type	Logic Error	Status	Fixed
Files	circuits/circuits/hash_to_field.circom		
Functions	Template ExpandMessageXMD		

The template `ExpandMessageXMD` computes the Sha256 hash of the following bit string `b_0 = sha256(Z_pad || msg || l_i_b_str || i2osp(0, 1) || DST_prime)` where `Z_pad` is the zero padding. The zero-padding should be an array of 64 zeros but it can in fact be set to many different values. This is because it is computed using the template I2OSP shown below:

```
template I2OSP(l) {
    signal input in;
    signal output out[l];

    var value = in;
    for (var i = l - 1; i >= 0; i--) {
        out[i] <-- value & 255;
        value = value \ 256;
    }

    signal acc[l];
    for (var i = 0; i < l; i++) {
        if (i == 0) {
            acc[i] <== out[i];
        } else {
            acc[i] <== 256 * acc[i-1] + out[i];
        }
    }

    acc[l-1] === in;
}
```

This template computes the bigint representation of `in` where `l` is the maximum number of digits and the base is 256. However, if `56l > log_2(p)` where `p` is the baby-jubjub prime, then we can have multiple representations for a given `in`. In this case, `l = 64` which is much larger than `56log_2(p)` In particular, if `in` is 0, then `out` can be the following:

```
[48, 100, 78, 114, 225, 49, 160, 41, 184, 80, 69, 182, 129, 129, 88, 93, 40, 51, 232,
     72, 121, 185, 112, 145, 67, 225, 245, 147, 240, 0, 0, 1]
```

This is the bigint representation of `p` and results in `acc[l-1] = 0`.

Impact An attacker can fill in this zero padding array with any bigint representation of a multiple of `p` . This allows attackers to map a given message to multiple field elements thereby increasing the surface area for forgeries.

Recommendation We suggest multiple fixes. We first suggest that the zero pad registers are explicitly set to zero rather than using `I2OSP`. This can be done as shown below:

```
component i2ospLibStr = I2OSP(2);
i2ospLibStr.in <== EXPANDED_LEN;

// b_0 = sha256(Z_pad || msg || l_i_b_str || i2osp(0, 1) || DST_prime)
var S256_0_INPUT_BYTE_LEN = R_IN_BYTES + MSG_LEN + 2 + 1 + DST_LEN + 1;
component sha0 = Sha256Bytes(S256_0_INPUT_BYTE_LEN);
for (var i = 0; i < S256_0_INPUT_BYTE_LEN; i++) {
    if (i < R_IN_BYTES) {
      sha0.in[i] <== 0; // THE FIX
  } else if (i < R_IN_BYTES + MSG_LEN) {
      sha0.in[i] <== msg[i - R_IN_BYTES];
  } else if (i < R_IN_BYTES + MSG_LEN + 2) {
      sha0.in[i] <== i2ospLibStr.out[i - R_IN_BYTES - MSG_LEN];
  } else if (i < R_IN_BYTES + MSG_LEN + 2 + 1) {
      sha0.in[i] <== 0;
  } else {
      sha0.in[i] <== dstPrime[i - R_IN_BYTES - MSG_LEN - 2 - 1];
  }
}
```

We also suggest that an assertion be added to `I2OSP` to prevent potential overflows. One assertion could be:

```
assert(l < 31);
```

This will prevent `acc[l-1]` from overflowing as `1acc[l-1] < 256^3 < p`.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.4 V-SUC-VUL-004: Sync committee can be rotated successfully with random public keys

Severity	Critical	**Commit**	f61894ea121d9b1b885
Type	Logic Error	**Status**	Fixed
Files	circuits/circuits/rotate.circom		
Functions	Template Rotate		

Template `Rotate` calculates the new `Poseidon` commitment based on both coordinates of the input public keys, `pubkeysBigInt` (second part of attached snippet). However, only the first coordinate of the public keys are used to verify the root of the next sync committee (first part of attached snippet). Therefore, an attacker can create a valid proof for `Rotate` by selecting random y coordinates for all public keys stored in `pubkeysBigInt`.

```
    /* VERIFY THE SSZ ROOT OF THE SYNC COMMITTEE */
    component sszSyncCommittee = SSZPhase0SyncCommittee();
    for (var i = 0; i < SYNC_COMMITTEE_SIZE; i++) {
        for (var j = 0; j < 48; j++) {
            sszSyncCommittee.pubkeys[i][j] <== pubkeysBytes[i][j];
        }
    }
    ...
    /* VERIFY THE POSEIDON ROOT OF THE SYNC COMMITTEE */
    component computePoseidonRoot = PoseidonG1Array(
        SYNC_COMMITTEE_SIZE,
        N,
        K
    );
    for (var i = 0; i < SYNC_COMMITTEE_SIZE; i++) {
        for (var j = 0; j < K; j++) {
            computePoseidonRoot.pubkeys[i][0][j] <== pubkeysBigInt[i][0][j];
            computePoseidonRoot.pubkeys[i][1][j] <== pubkeysBigInt[i][1][j];
        }
    }
```

Impact Such an attack can render the `step` function of `telepathy` unusable for a whole `period` and lock user funds indefinitely, since the smart contract allows the commitment to be updated only once per period. At an extreme, this attack can turn into a DoS if an attacker is able to maliciously rotate the committee on every period.

Recommendation We recommend computing the `Poseidon` commitment by using only the x coordinates of the committee's public keys.

Developer Response Our teams iterated over this issue multiple times after the official end of our audit. Our initial recommendation for committing only to the x-coordinates of each key required an additional subgroup check for every committee member, which resulted in a prohibitively expensive circuit. After several interactions with the Succinct Labs team, we discovered that the most significant bits of the x-coordinate encode the sing of y-coordinate of

every public key (recall that elliptic curves are symmetric on the x-axis). This opened the door to the following cheaper and straightforward check: First, recover the sign from the x-coordinate of the provided key. Then, simply check that the sign agrees with the y-coordinate of the key and that the (x, y) lies on the BLS curve. The Succinct Labs team implemented this logic in their final version of their code.

4.1.5 V-SUC-VUL-005: Template LongToShortNoEndCarry is incomplete for some instantiations

Severity	Medium	**Commit**	f61894ea121d9b1b885
Type	Logic Error	**Status**	Fixed
Files	circuits/pairing/bigint.circom		
Functions	Template LongToShortNoEndCarry		

Template `LongToShortNoEndCarry` implementation performs a split case on one of its parameters, `k`, in order to calculate its output signal. However, when `k` is either 1 or 2, `out[k]` is never set.

```
template LongToShortNoEndCarry(n, k) {
    assert(n <= 126);
    signal input in[k];
    signal output out[k+1];

    var split[k][3];
    for (var i = 0; i < k; i++) {
        split[i] = SplitThreeFn(in[i], n, n, n);
    }

    var carry[k];
    carry[0] = 0;
    out[0] <-- split[0][0];
    if (k > 1) {
        var sumAndCarry[2] = SplitFn(split[0][1] + split[1][0], n, n);
        out[1] <-- sumAndCarry[0];
        carry[1] = sumAndCarry[1];
    }
    if (k > 2) {
        for (var i = 2; i < k; i++) {
            var sumAndCarry[2] = SplitFn(split[i][0] + split[i-1][1] + split[i-2][2]
    + carry[i-1], n, n);
            out[i] <-- sumAndCarry[0];
            carry[i] = sumAndCarry[1];
        }
        out[k] <-- split[k-1][1] + split[k-2][2] + carry[k-1];
    }

    component outRangeChecks[k+1];
    for (var i = 0; i < k+1; i++) {
        outRangeChecks[i] = Num2Bits(n);
        outRangeChecks[i].in <== out[i];
    }

    signal runningCarry[k];
    component runningCarryRangeChecks[k];
    runningCarry[0] <-- (in[0] - out[0]) / (1 << n);
    runningCarryRangeChecks[0] = Num2Bits(n + log_ceil(k));
    runningCarryRangeChecks[0].in <== runningCarry[0];
    runningCarry[0] * (1 << n) === in[0] - out[0];
    for (var i = 1; i < k; i++) {
```

```
41          runningCarry[i] <-- (in[i] - out[i] + runningCarry[i-1]) / (1 << n);
42          runningCarryRangeChecks[i] = Num2Bits(n + log_ceil(k));
43          runningCarryRangeChecks[i].in <== runningCarry[i];
44          runningCarry[i] * (1 << n) === in[i] - out[i] + runningCarry[i-1];
45      }
46      runningCarry[k-1] === out[k];
47  }
```

Impact Instantiations of this template where `k = 1` or `k = 2` will lead to incorrect witnesses that won't be verifiable.

Recommendation Set `out[k]` to the expected value for the aforementioned cases.

Developer Response The developers partially fixed this issue by blocking invalid instantiations in this commit.

4.1.6 V-SUC-VUL-006: ModSumFour is Incorrect

Severity	Medium	Commit	f61894ea121d9b1b885
Type	Logic Error	Status	Fixed
Files	circuits/pairing/bigint.circom		
Functions	Template ModSumFour		

`ModSumFour`, shown below, incorrectly asserts that `n + 2 <= 253`. This allows `n` to be large enough such that `a + b + c + d` overflows the prime. For example, `a, b, c, d` can be `2^252-1` and so their sum would be `542^2 - 4` which is larger than the baby-jubjub prime.

```
template ModSumFour(n) {
    assert(n + 2 <= 253);
    signal input a;
    signal input b;
    signal input c;
    signal input d;
    signal output sum;
    signal output carry;

    component n2b = Num2Bits(n + 2);
    n2b.in <== a + b + c + d;
    carry <== n2b.out[n] + 2 * n2b.out[n + 1];
    sum <== a + b + c + d - carry * (1 << n);
}
```

Impact Currently no other templates use `ModSumFour` but if it was to be used this needs to be changed.

Recommendation We recommend a fix like the following:

```
template ModSumFour(n) {
    assert(n + 3 <= 253);
  signal input a;
  signal input b;
  signal input c;
  signal input d;

  component n2b = Num2Bits(n + 3);
  n2b.in <== a + b + c + d;
  carry <== n2b.out[n] + 2 * n2b.out[n + 1] + 4*n2b.out[n+2];
  sum <== a + b + c + d - carry * (1 << n);
}
```

Developer Response The developers removed this unused template in this commit.

4.1.7 V-SUC-VUL-007: Template G1AddMany assumes that no intermediate sum is equal to the point at infinity.

Severity	Medium	**Commit**	f61894ea121d9b1b885
Type	Data Validation	**Status**	Fixed
Files	circuits/circuits/bls.circom		
Functions	Template G1AddMany		

The template `G1AddMany` adds all the public keys used to sign the block header. It aggregates the keys using the template `G1Add` which in turn uses`EllipticCurveAddUnequal`. The latter template assumes that the inputs are unequal and do not sum to the point at infinity, however there is no check making sure the inputs are unequal and the sum cannot be the point at infinity.

Impact Since EllipticCurveAddUnequal assumes that the inputs are not points at infinity, it is possible that a correctly signed header is rejected because at some point in the computation, the public keys end up adding to infinity. Similarly in case both input are equal. Moreover, it is possible that `EllipticCurveAddUnequal` can produce a point on the curve (since it is undefined if the sum actually adds to the point at infinity) and thus allow a bogus header to get published.

Recommendation We recommend the template `G1Add` add a check to make sure the two points do not sum to infinity. Such a check would make sure the x coordinates of the public keys are not the same.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.8 V-SUC-VUL-008: VerifySyncCommittee should make sure that not all the aggregation bits are 0

Severity	Medium	Commit	f61894ea121d9b1b885
Type	Data Validation	Status	Fixed
Files	circuits/circuits/sync_committee.circom		
Functions	Template VerifySyncCommittee		

If all the aggregation bits are zero, then `G1AddArray` returns the point `(0, 0)`in the circuit `VerifySyncCommittee`.

Impact Currently, this will not cause issues because `(0, 0)` is not on the curve but if the signature scheme changes and `(0, 0)` is on the curve then someone can publish an unsigned fake header by setting all aggregation bits to 0.

Recommendation We recommend adding the following lightweight check to `VerifySyncCommittee` .

```
    /* RANGE CHECK AGGREGATION BITS */
    var sum = 0;
    for (var i = 0; i < SYNC_COMMITTEE_SIZE; i++) {
        aggregationBits[i] * (aggregationBits[i] - 1) === 0;
        sum += aggregationBits[i];
    }
    component zeroCheck = IsZero();
    zeroCheck.in <== sum;
    zeroCheck.out === 0;
```

Developer Response The developers acknowledged and fixed the issue at this location.

4.1.9 V-SUC-VUL-009: Templates Split and SplitThree have non-deterministic instantiations

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Data Validation	**Status**	Fixed
Files	circuits/pairing/bigint.circom		
Functions	Templates Split and SplitThree		

Templates Split and SplitThree can have non-deterministic instantiations due to missing parameter validation. Specifically, template `Split` does not restrict parameter `m`, which is used to instantiate a `Num2Bits` template from the circom library. If `m` is greater than 253, then the output signals of `Split` will not be uniquely identified by input signals. Similarly template `SplitThree` does not put any restrictions on arguments `m` and `k`.

```
template Split(n, m) {
    assert(n <= 126);
    ...
    component n2b_big = Num2Bits(m);
    ...
}

template SplitThree(n, m, k) {
    assert(n <= 126);
    ...
    component n2b_medium = Num2Bits(m);
    n2b_medium.in <== medium;
    component n2b_big = Num2Bits(k);
    ...
}
```

Counterexample A counterexample has been constructed using Veridise's tool Picus.

Impact Client code may wrongly instantiate these templates and also become non-deterministic.

Recommendation Template `Split`: consider adding assertion `n + m < 254`.

Template `SplitThree`: consider adding assertion `n + m + k < 254`.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.10 V-SUC-VUL-010: Implicit assumption in BigMult

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Data Validation	**Status**	Fixed
Files	circuits/pairing/bigint.circom		
Functions	Template BigMult		

Template `BigMult` implicitly assumes that `k <= 2^n`. As shown in the attached code snippet, `BigMult` is a composition of circuits `BigMultShortLong` and `LongToShortNoEndCarry`. The output of `BigMultShortLong` consists of `2k - 1` registers with each register containing integers in the range `[0, 2^(2n + log(k))`. This output is then passed as input to `LongToShortNoEndCarry` circuit, which converts the output of `BigMultShortLong` to proper integer representation. Additionally, circuit `LongToShortNoEndCarry` assumes that each register of the input fits in `3n` bits. Therefore, from the above it follows that `2n + log(k) <= 3n`. However, this is not enforced by `BigMult`.

```
    var LOGK = log_ceil(k);
    component mult = BigMultShortLong(n, k, 2*n + LOGK);
    for (var i = 0; i < k; i++) {
        mult.a[i] <== a[i];
        mult.b[i] <== b[i];
    }

    // no carry is possible in the highest order register
    component longshort = LongToShortNoEndCarry(n, 2 * k - 1);
    for (var i = 0; i < 2 * k - 1; i++) {
        longshort.in[i] <== mult.out[i];
    }
```

Impact Improper instantiation of `BigMult` may lead to wrong output signals.

Recommendation Please consider adding an assertion in `BigMult`.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.11 V-SUC-VUL-011: BigMod does not work for all instantiations

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Fixed
Files	circuits/pairing/bigint.circom		
Functions	Template BigMod		

If the template is instantiated with `k >= 50`, then the following excerpt from BigMod is incorrect:

```
    div[k] <-- longdiv[0][k];
    component div_range_checks[k + 1];
    for (var i = 0; i <= k; i++) {
       div_range_checks[i] = Num2Bits(n);
       div_range_checks[i].in <== div[i];
    }
```

This is because the second dimension of `longdiv` is of size 50 and if `k > 50` this will result in an index out of bounds during witness generation.

Impact No impact to the current code base but any users of the template should be careful.

Recommendation Add an assertion `assert(k < 50)` at the beginning of the method.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.12 V-SUC-VUL-012: BigMod2 does not work for all instantiations

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Fixed
Files	circuits/pairing/bigint.circom		
Functions	Template BigMod2		

If the template is instantiated with `k > 50` or `m - k >= 50`, then the following excerpt from BigMod2 is incorrect:

```
    var longdiv[2][50] = long_div2(n, k, m-k, a, b);
    for (var i = 0; i < k; i++) {
        mod[i] <-- longdiv[1][i];
    }
    for (var i = 0; i <= m-k; i++) {
        div[i] <-- longdiv[0][i];
    }
```

This is because the second dimension of `long_div2` is of size 50 and if any of the above conditions hold this will result in an index out of bounds during witness generation.

Impact No impact to the current code base but any users of the template should be careful.

Recommendation Add an assertion `assert(k <= 50 && m - k < 50)` at the beginning of the method.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.13 V-SUC-VUL-013: BigModInv does not work for all instantiations

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Fixed
Files	circuits/pairing/bigint.circom		
Functions	Template BigModInv		

The template `BigModInv` does not work for all instantiations of `k`. In particular, if `k > 50` then this can cause witness generation to fail. In particular, the following excerpt from the template is buggy:

```
    // length k
    var inv[50] = mod_inv(n, k, in, p);
    for (var i = 0; i < k; i++) {
        out[i] <-- inv[i];
    }
```

This is because the access `inv[i]` can be out-of-bounds when `k >= 50`

Impact If a client instantiates this template with `k >= 50` then witness generation will fail.

Recommendation Since `BigModInv` calls `BigMod` we believe adding the apprioriate `assert(k < 50)` to the beginning of `BigMod` should fix this issue as well.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.14 V-SUC-VUL-014: Implicit Assumption in BigModInv

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Fixed
Files	circuits/pairing/bigint.circom		
Functions	Template BigModInv		

`BigModInv` is only correct if `p` represents a prime number. This assumption is not made explicit in the comments.

Impact Any clients who do not pass in a prime `p` will not get a well defined output

Recommendation Add a comment saying `// p represents a prime`.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.15 V-SUC-VUL-015: Constant 512 should be replaced with SYNC_COMMITTEE_SIZE

Severity	Low	Commit	f61894ea121d9b1b885
Type	Maintainability	Status	Fixed
Files	circuits/circuits/bls.circom		
Functions	Template G1AddMany		

The template `G1AddMany` shown below takes the parameter `SYNC_COMMITTEE_SIZE`. The variable `BATCH_SIZE` should use `SYNC_COMMITTEE_SIZE` instead of `512`

```
template G1AddMany(SYNC_COMMITTEE_SIZE, LOG_2_SYNC_COMMITTEE_SIZE, N, K) {
    signal input pubkeys[SYNC_COMMITTEE_SIZE][2][K];
    signal input bits[SYNC_COMMITTEE_SIZE];
    signal output out[2][K];

    component reducers[LOG_2_SYNC_COMMITTEE_SIZE];
    for (var i = 0; i < LOG_2_SYNC_COMMITTEE_SIZE; i++) {
        var BATCH_SIZE = 512 \ (2 ** i);
        reducers[i] = G1Reduce(BATCH_SIZE, N, K);
        for (var j = 0; j < BATCH_SIZE; j++) {
            if (i == 0) {
```

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.16 V-SUC-VUL-016: Fp2Invert does not work for all valid instantiations

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Fixed
Files	circuits/pairing/bigint.circom		
Functions	Template Fp2Invert		

If the template is instantiated with `k > 50`, then the following excerpt from `Fp2Invert` is incorrect:

```
1    var inverse[2][50] = find_Fp2_inverse(n, k, in, p); // 2 x 50, only 2 x k
     relevant
2    for (var i = 0; i < 2; i ++) {
3        for (var j = 0; j < k; j ++) {
4            out[i][j] <-- inverse[i][j];
5        }
6    }
```

This is because the second dimension of `pow2nk` is of size 50 and if `k > 50` this will result in an index out of bounds during witness generation.

Impact No impact to the current code base but any users of the template should be careful.

Recommendation Add an assertion `assert(k <= 50)` at the beginning of the method.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.17 V-SUC-VUL-017: SignedFp2Divide does not work for all valid instantiations

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Fixed
Files	circuits/pairing/bigint.circom		
Functions	Template SignedFp2Divide		

If the template is instantiated with `k > 50`, then the following excerpts from `SignedFp2Divide` are incorrect:

```
var a_mod[2][50];
var b_mod[2][50];
for(var eps=0; eps<2; eps++){
    // 2^{overflow} <= 2^{n*ceil(overflow/n)}
    var temp[2][50] = get_signed_Fp_carry_witness(n, k, ma, a[eps], p);
    a_mod[eps] = temp[1];
    temp = get_signed_Fp_carry_witness(n, k, mb, b[eps], p);
    b_mod[eps] = temp[1];
}

// precompute 1/b
var b_inv[2][50] = find_Fp2_inverse(n, k, b_mod, p);
// precompute a/b
var out_var[2][50] = find_Fp2_product(n, k, a_mod, b_inv, p);

...

var m = max( mb + k, ma );
// get mult = out * b = p*X' + Y'
var XY[2][2][50] = get_signed_Fp2_carry_witness(n, k, m, mult.out, p); // total value
     is < 2^{nk} * 2^{n*k + overflowb - n + 1}
// get a = p*X' + Y'
var XY1[2][2][50] = get_signed_Fp2_carry_witness(n, k, m, a, p); // same as above, m
    extra registers enough
```

This is because all of these arrays have a dimension of size 50 and if `k > 50` this will result in an index out of bounds during witness generation.

Impact No impact to the current code base but any users of the template should be careful.

Recommendation Add an assertion `assert(k <= 50)` at the beginning of the method.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.18 V-SUC-VUL-018: Constant 32 should be replaced with B_IN_BYTES in ExpandMessageXMD

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Fixed
Files	circuits/circuits/hash_to_field.circom		
Functions	Template ExpandMessageXMD		

In the template `ExpandMessageXMD` the constant 32 is used in many places instead of the variable `B_IN_BYTES`. The following code snippet is one example:

```
    for (var i = 0; i < S256S_0_INPUT_BYTE_LEN; i++) {
        if (i < 32) {
            s256s[0].in[i] <== sha0.out[i];
        } else if (i < 32 + 1) {
            s256s[0].in[i] <== 1;
        } else {
            s256s[0].in[i] <== dstPrime[i - 32 - 1];
        }
    }
```

Impact If the value of `B_IN_BYTES` changes all these usages of 32 need to be changed as well. Failure to change one of them will break the functional correctness of the template.

Recommendation We recommend replacing usages of the constant 32 with `B_IN_BYTES`.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.19 V-SUC-VUL-019: PrimeReduce does not work for all valid instantiations

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Fixed
Files	circuits/pairing/bigint.circom		
Functions	Template PrimeReduce		

If the template is instantiated with `k > 50`, then the following excerpt from `PrimeReduce` is incorrect:

```
    e[0] = n;
    var pow2n[50] = mod_exp(n, k, two, p, e);
    e[0] = k;
    assert(k < (1<<n) );
    var pow2nk[50] = mod_exp(n, k, pow2n, p, e);
```

This is because `pow2nk` is of size 50 and if `k > 50` this will result in an index out of bounds during witness generation.

Impact No impact to the current code base but any users of the template should be careful.

Recommendation Add an assertion `assert(k <= 50)` at the beginning of the method.

Developer Response The developers acknowledged and fixed the issue at this location.

4.1.20 V-SUC-VUL-020: SSZPhase0SyncCommittee should use template parameters

Severity	Low	Commit	f61894ea121d9b1b885
Type	Maintainability	Status	Fixed
Files	circuits/circuits/ssz.circom		
Functions	Template SSZPhase0SyncCommittee		

The template `SSZPhase0SyncCommittee` hard codes the number of public keys to be 512 but it should take a parameter `SYNC_COMMITTEE_SIZE`.

Impact If the `SYNC_COMMITTEE_SIZE` changes, the array `pubkeys` needs to be redeclared and the loop condition here has to be changed.

Recommendation We recommend the following fix:

```
template SSZPhase0SyncCommittee(SYNC_COMMITTEE_SIZE, LOG_SYNC_COMMITTEE_SIZE) {
        assert(SYNC_COMMITTEE_SIZE > 0);
        assert(2**LOG_SYNC_COMMITTEE_SIZE == SYNC_COMMITTEE_SIZE);
    signal input pubkeys[SYNC_COMMITTEE_SIZE][48];
    signal input aggregatePubkey[48];
    signal output out[32];

    component sszPubkeys = SSZArray(SYNC_COMMITTEE_SIZE*64, LOG_SYNC_COMMITTEE_SIZE
    +1);
    for (var i = 0; i < SYNC_COMMITTEE_SIZE; i++) {
        for (var j = 0; j < 64; j++) {
            if (j < 48) {
                sszPubkeys.in[i * 64 + j] <== pubkeys[i][j];
            } else {
                sszPubkeys.in[i * 64 + j] <== 0;
            }
        }
    }

    component sszAggregatePubkey = SSZArray(64, 1);
    for (var i = 0; i < 64; i++) {
        if (i < 48) {
            sszAggregatePubkey.in[i] <== aggregatePubkey[i];
        } else {
            sszAggregatePubkey.in[i] <== 0;
        }
    }

    component hasher = Sha256Bytes(64);
    for (var i = 0; i < 64; i++) {
        if (i < 32) {
            hasher.in[i] <== sszPubkeys.out[i];
        } else {
            hasher.in[i] <== sszAggregatePubkey.out[i - 32];
        }
```

```
35      }
36
37      for (var i = 0; i < 32; i++) {
38          out[i] <== hasher.out[i];
39      }
40  }
```

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.21 V-SUC-VUL-021: SSZLayer assumes numBytes 64

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Data Validation	**Status**	Fixed
Files	circuits/circuits/ssz.circom		
Functions	Template SSZLayer		

The template `SSZLayer` , shown below, implicitly assumes parameter `numBytes` 64. However there is no assertion enforcing this.

```
template SSZLayer(numBytes) {
    signal input in[numBytes];
    signal output out[numBytes\ 2];

    var numPairs = numBytes \ 64;
    component hashers[numPairs];

    for (var i = 0; i < numPairs; i++) {
        hashers[i] = Sha256Bytes(64);
        for (var j = 0; j < 64; j++) {
            hashers[i].in[j] <== in[i * 64 + j];
        }
    }

    for (var i = 0; i < numPairs; i++) {
        for (var j = 0; j < 32; j++) {
            out[i * 32 + j] <== hashers[i].out[j];
        }
    }
}
```

Impact If `numBytes` < 64, then `numPairs = 0` and both loops are skipped. Thus `out` will be completely underconstrained.

This has no impact in the codebase currently since the calling templates force `numBytes` 64 . However, if this template is used somewhere else and `numBytes` is < 64 then it can result in an underconstrained bug.

Recommendation We recommend adding an assertion `numBytes` 64

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.22 V-SUC-VUL-022: SSZRestoreMerkelRoot signal value is unused

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Fixed
Files	circuits/circuits/ssz.circom		
Functions	Template SSZRestoreMerkelRoot		

The intermediate signal `value` is not used in template `SSZRestoreMerkelRoot`.

Impact This should not affect the rest of the circuit but affects readability.

Recommendation We recommend removing that signal.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.23 V-SUC-VUL-023: EllipticCurveAddUnequal doesn't work for all instantiations

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Fixed
Files	circuits/pairing/curve.circom		
Functions	Template EllipticCurveAddUnequal		

If the template is instantiated with `k >= 50`, then the following excerpt from `EllipticCurveAddUnequal` is incorrect:

```
    signal output out[2][k];
    // Ignored multiple lines
    for(var i = 0; i < k; i++){
            out[0][i] <-- x3[i];
            out[1][i] <-- y3[i];
    }
```

This is because the second dimension of `out` is of size 50 and if `k > 50` this will result in an index out of bounds during witness generation.

Impact No impact on the current code base but any users of the template should be careful.

Recommendation Add an assertion `assert(k < 50)` at the beginning of the method.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.24 V-SUC-VUL-024: Fp12Invert doesn't work for all instantiations

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Fixed
Files	circuits/pairing/fp12.circom		
Functions	Template Fp12Invert		

If the template is instantiated with `k >= 50`, then the following excerpt from `Fp12Invert` is incorrect:

```
    var inverse[6][2][50] = find_Fp12_inverse(n, k, p, in); // 6 x 2 x 50, only 6 x 2
     x k relevant
    for (var i = 0; i < 6; i ++) {
        for (var j = 0; j < 2; j ++) {
            for (var m = 0; m < k; m ++) {
                out[i][j][m] <-- inverse[i][j][m];
            }
        }
    }
```

This is because the third dimension of `inverse` is of size 50 and if `k > 50` this will result in an index out of bounds during witness generation.

Impact No impact on the current code base but any users of the template should be careful.

Recommendation Add an assertion `assert(k < 50)` at the beginning of the method.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.25 V-SUC-VUL-025: SignedCheckCarryModToZero doesn't work for all instantiations

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Fixed
Files	circuits/pairing/fp.circom		
Functions	Template SignedCheckCarryModToZero		

If the template is instantiated with `k >= 50`, then the following excerpt from `SignedCheckCarryModToZero` is incorrect:

```
    var Xvar[2][50] = get_signed_Fp_carry_witness(n, k, m, in, p);
    component X_range_checks[m];

    for(var i=0; i<m; i++){
        X[i] <-- Xvar[0][i];
        X_range_checks[i] = Num2Bits(n+1);
        X_range_checks[i].in <== X[i] + (1<<n); // X[i] should be between [-2^n, 2^n)
    }
```

This is because the second dimension of `Xvar` is of size 50 and if `m > 50` this will result in an index out of bounds during witness generation.

Impact No impact on the current code base but any users of the template should be careful.

Recommendation Add an assertion `assert(m < 50)` at the beginning of the method.

Developer Response The developers acknowledged and fixed the issue at this location.

4.1.26 V-SUC-VUL-026: long_add4 doesn't work for all invocations

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Data Validation	**Status**	Fixed
Files	circuits/pairing/bigint$_{f}unc.circom$		
Functions	Function long$_{a}dd4$		

If the function is called with `k >= 50`, then the following excerpt from `long_add4` is incorrect:

```
function long_add4(n, k, a, b, c, d){
    var carry = 0;
    var sum[50];
    for(var i=0; i < k; i++){
        var sumAndCarry[2] = SplitFn(a[i] + b[i] + c[i] + d[i] + carry, n, n);
        sum[i] = sumAndCarry[0];
        carry = sumAndCarry[1];
    }
    sum[k] = carry;
    return sum;
}
```

This is because the first dimension of `sum` is of size 50 and if `k > 50` this will result in an index out of bounds during execution.

Impact No impact on the current code base but any users of the function should be careful.

Recommendation Add an assertion `assert(k < 50)` at the beginning of the method.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.27 V-SUC-VUL-027: long_add_unequal doesn't work for all invocations

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Data Validation	**Status**	Fixed
Files	circuits/pairing/bigint_func.circom		
Functions	Function long_add_unequal		

If the function is called with `k1 >= 50`, then the following excerpt from `long_add_unequal` is incorrect:

```
function long_add_unequal(n, k1, k2, a, b){
    var carry = 0;
    var sum[50];
    for(var i=0; i<k1; i++){
        if (i < k2) {
            var sumAndCarry[2] = SplitFn(a[i] + b[i] + carry, n, n);
            sum[i] = sumAndCarry[0];
            carry = sumAndCarry[1];
        } else {
            var sumAndCarry[2] = SplitFn(a[i] + carry, n, n);
            sum[i] = sumAndCarry[0];
            carry = sumAndCarry[1];
        }
    }
    sum[k1] = carry;
    return sum;
}
```

This is because the first dimension of `sum` is of size 50 and if `k1 >= 50` this will result in an index out of bounds during execution.

Impact No impact on the current code base but any users of the function should be careful.

Recommendation Add an assertion `assert(k1 < 50)` at the beginning of the method.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.28 V-SUC-VUL-028: SignedFpCarryModP doesn't work for all instantiations

Severity	Low	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Fixed
Files	circuits/pairing/fp.circom		
Functions	Template SignedFpCarryModP		

If the template is instantiated with `k >= 50`, then the following excerpt from `SignedFpCarryModP` is incorrect:

```
    var Xvar[2][50] = get_signed_Fp_carry_witness(n, k, m, in, p);
    component X_range_checks[m];
    component range_checks[k];
    //component lt = BigLessThan(n, k);

    for(var i=0; i<k; i++){
        out[i] <-- Xvar[1][i];
        range_checks[i] = Num2Bits(n);
        range_checks[i].in <== out[i];
        //lt.a[i] <== out[i];
        //lt.b[i] <== p[i];
    }
```

This is because the second dimension of `Xvar` is of size 50 and if `k >= 50` this will result in an index out of bounds during witness generation.

Impact No impact on the current code base but any users of the template should be careful.

Recommendation Add an assertion `assert(k < 50)` at the beginning of the method.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.29 V-SUC-VUL-029: Assumptions in long_add_unequal should be asserted explicitly

Severity	Low	Commit	f61894ea121d9b1b885
Type	Data Validation	Status	Fixed
Files	circuits/pairing/bigint_func.circom		
Functions	Function long_add_unequal		

`k1 > k2` should assert explicitly according to the comment here.

```
1   // n bits per register
2   // a has k1 registers
3   // b has k2 registers
4   // assume k1 > k2
5   // output has k1+1 registers
6   function long_add_unequal(n, k1, k2, a, b){
7       var carry = 0;
8       var sum[50];
9       for(var i=0; i<k1; i++){
10          if (i < k2) {
11              var sumAndCarry[2] = SplitFn(a[i] + b[i] + carry, n, n);
12              sum[i] = sumAndCarry[0];
13              carry = sumAndCarry[1];
14          } else {
15              var sumAndCarry[2] = SplitFn(a[i] + carry, n, n);
16              sum[i] = sumAndCarry[0];
17              carry = sumAndCarry[1];
18          }
19      }
20      sum[k1] = carry;
21      return sum;
22  }
```

Impact No impact on the current code base but any users of the function should be careful.

Recommendation Add an assertion `assert(k1 > k2)` at the beginning of the method.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.30 V-SUC-VUL-030: Intermediate Signal add.out is under constrained in BigSubModP

Severity	Warning	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Won't Fix
Files	circuits/pairing/bigint.circom		
Functions	Template BigSubModP		

In the circuit `BigSubModP`, shown below, the intermediate signals `add.out` are underconstrained.

```
// calculates (a - b) % p, where a, b < p
// note: does not assume a >= b
template BigSubModP(n, k){
    assert(n <= 252);
    signal input a[k];
    signal input b[k];
    signal input p[k];
    signal output out[k];
    component sub = BigSub(n, k);
    for (var i = 0; i < k; i++){
        sub.a[i] <== a[i];
        sub.b[i] <== b[i];
    }
    signal flag;
    flag <== sub.underflow;
    component add = BigAdd(n, k);
    for (var i = 0; i < k; i++){
        add.a[i] <== sub.out[i];
        add.b[i] <== p[i];
    }
    signal tmp[k];
    for (var i = 0; i < k; i++){
        tmp[i] <== (1 - flag) * sub.out[i];
        out[i] <== tmp[i] + flag * add.out[i];
    }
}
```

If `flag = 0` note that `add.out` is not included in any assertion. Similarly, if `flag = 1` then `add.out[k]` is not included in any constrained.

Impact We think that there is no apparent vulnerability due to this, however we think it is worth mentioning as a warning since we can't prove that it is not possible to exploit.

Recommendation Add a comment noting that `add[k]`is unrestricted in the implementation.

Developer Response This mainly affects the circom-pairing library maintainability. Thus, it is not crucial for fixing it in the context of this project.

4.1.31 V-SUC-VUL-031: HashToField is under constrained

Severity	Warning	Commit	f61894ea121d9b1b885
Type	Logic Error	Status	Fixed
Files	circuits/circuits/hash_to_field.circom		
Functions	Template HashToField		

The template `HashToField` is underconstrained as the function `SignedFpCarryModP` is underconstrained. This would normally be okay since `CoreVerifyPublicKeyG1` checks that the field element is less than the prime; however, that check is incorrect.

Developer Response This is implicitly fixed by the fix of the `CoreVerifyPubkeyG1` data validation issue.

4.1.32 V-SUC-VUL-032: Make all depth constants the floorlog2 of their index constants

Severity	Warning	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Won't Fix
Files	circuits/circuits/constants.circom		
Functions	Functions that return depth of an index.		

There are at least three index constants which have corresponding depth constants: `FinalizedHeaderIndex` , `ExecutionStateRootIndex`, and `SyncCommitteeIndex` . Since the depth constants should always be the floor of the `log_2` of the index constants, we recommend changing the functions which get the depth constants to compute the `floorlog2` of the corresponding index constant rather than returning a hardcoded constant. This way if one of the index constants is changed, the depth constant should not need to be updated .

Impact If the index constant is changed but depth constant isn't then the implementation will not be correct.

Recommendation We recommend writing a function that computes `floorlog2` and making the functions which get the depth constants use it on their corresponding index functions. For example, we recommend changing `getFinalizedHeaderDepth` to:

```
    function getFinalizedHeaderDepth() {
        return log2_floor(getFinalizedHeaderIndex());
    }
```

Developer Response The developers acknowledged this issue but chose not to implement our recommendation at this point.

4.1.33 V-SUC-VUL-033: Parameter N is unused in template PoseidonG1Array

Severity	Warning	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Won't Fix
Files	circuits/circuits/poseidon.circom		
Functions	Template PoseidonG1Array		

The template PoseidonG1Array shown below takes a parameter `N` which is not used.

```
template PoseidonG1Array(LENGTH, N, K) {
    signal input pubkeys[LENGTH][2][K];
    signal output out;

    component hasher = PoseidonFieldArray(LENGTH * 2 * K);
    for (var i = 0; i < LENGTH; i++) {
        for (var j = 0; j < K; j++) {
            for (var l = 0; l < 2; l++) {
                hasher.in[(i * K * 2) + (j * 2) + l] <== pubkeys[i][l][j];
            }
        }
    }
    out <== hasher.out;
}
```

Impact The main impact is readability. It would be good to have some documentation for why `N` is taken as a parameter even though it is unused

Recommendation We recommend that `N` is removed or a comment is added indicating the parameter is unused and why it is important.

Developer Response This is a minor issue that does not affect the overall application.

4.1.34 V-SUC-VUL-034: Input signal aggregatePubkeyBigInt can be dropped from template Rotate without comprimising security

Severity	Info	**Commit**	f61894ea121d9b1b885
Type	Optimization	**Status**	Fixed
Files	circuits/circuits/rotate.circom		
Functions	Template Rotate		

The template `Rotate` takes a private input signal `aggregatePubkeyBigInt` which is only used to validate that the conversion of `aggregatePubkey` bytes to bigints is done correctly. However, this check is not necessary since a malicious attacker can always compute `G1BytesToBigInt` locally for a given `aggregatePubkey` and use the corresponding output when generating the proof.

Impact The input and code which uses it makes the implementation unnecessarily complex.

Recommendation We recommend removing this input along with all code using it as this would reduce the number of constraints and simplify the implementation as a whole.

Developer Response The developers acknowledged and fixed the issue in this commit.

4.1.35 V-SUC-VUL-035: Duplicate Code Snippet

Severity	Info	**Commit**	f61894ea121d9b1b885
Type	Maintainability	**Status**	Won't Fix
Files	circuits/circuits/inputs.circom		
Functions	Template SerializeLightClientStepInputs		

The code that converts signals to little endian format has been duplicated inside `SerializeLightClientStepInputs`.

```
    /* participationLE = toLittleEndian(participation) */
    component bitify0 = Num2Bits_strict();
    bitify0.in <== participation;
    component byteify0[32];
    for (var i = 0; i < 32; i++) {
        byteify0[i] = Bits2Num(8);
        for (var j = 0; j < 8; j++) {
            if (i*8+j < TRUNCATED_SHA256_SIZE) {
                byteify0[i].in[j] <== bitify0.out[i*8+j];
            } else {
                byteify0[i].in[j] <== 0;
            }
        }
    }

    ...

    /* syncCommitteePoseidonLE = toLittleEndian(syncCommitteePoseidon) */
    component bitify1 = Num2Bits_strict();
    bitify1.in <== syncCommitteePoseidon;
    component byteify1[32];
    for (var i = 0; i < 32; i++) {
        byteify1[i] = Bits2Num(8);
        for (var j = 0; j < 8; j++) {
            if (i*8+j < 254) {
                byteify1[i].in[j] <== bitify1.out[i*8+j];
            } else {
                byteify1[i].in[j] <== 0;
            }
        }
    }
```

Impact This can create maintainability issues in the future.

Recommendation We recommend creating a new template for little endian conversion and use it in the above locations.

Developer Response The developers acknowledged this issue but chose not to implement our recommendation at this point.

4.1.36 V-SUC-VUL-036: Gratuitous constraints in SerializeLightClientStepInputs

Severity	Info	**Commit**	f61894ea121d9b1b885
Type	Optimization	**Status**	Won't Fix
Files	circuits/circuits/input.circom		
Functions	Template SerializeLightClientStepInputs		

Template `SerializeLightClientStepInputs` converts the result of `sha3` into bits unnecessarily. Template `Sha256`, which is used internally in `Sha256Bytes`, returns the hash in bit format. That means the result of `Sha256` is first converted to bytes and then back to bits again.

```
    /* h = sha256(h, syncCommitteePoseidonLE) */
    component sha3 = Sha256Bytes(64);
    for (var i = 0; i < 32; i++) {
        sha3.in[i] <== sha2.out[i];
        sha3.in[32+i] <== byteify1[i].out;
    }

    component bitifiers[32];
    for (var i = 0; i < 32; i++) {
        bitifiers[i] = Num2Bits(8);
        bitifiers[i].in <== sha3.out[i];
    }
```

Impact This adds unnecessary constraints in a system with millions of constraints.

Recommendation We recommend replacing `Sha256Bytes` with two separate templates in `circuits/circuits/sha256.circom`: 1. `Sha256BytesToBytes`, which will be equivalent to the current `Sha256Bytes` template and 2. `Sha256BytesToBits`, which will omit the conversion of `Sha256`'s result to bytes.

Developer Response The developers acknowledged this issue but chose not to implement our recommendation at this point.

