Polygon Labs Iden3 Circuits

Security Assessment

May 3, 2024

Prepared for:
Oleksandr Brezniev
Polygon Labs

Prepared by: **Tjaden Hess and Scott Arciszewski**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineering director was associated with this project:

Jim Miller, Engineering Director, Cryptography
james.miller@trailofbits.com

The following consultants were associated with this project:

Tjaden Hess, Consultant
tjaden.hess@trailofbits.com

Scott Arciszewski, Consultant
scott.arciszewski@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
March 11, 2024	Pre-project kickoff call
March 19, 2024	Status update meeting #1
March 22, 2024	Delivery of report draft
March 22, 2024	Report readout meeting
May 3, 2024	Delivery of comprehensive report with fix review appendix

Executive Summary

Engagement Overview

Polygon Labs engaged Trail of Bits to review the security of the Iden3 Circom circuits. Iden3 is an anonymous credential protocol that uses non-interactive zero-knowledge (ZK) proofs.

One consultant conducted the review, with another consultant shadowing, from March 11 to March 22, 2024, for a total of two engineer-weeks of effort. Our testing efforts focused on checking the Circom circuits for missing constraints and validating the security and privacy of the overall protocol. With full access to source code and documentation, we performed static and dynamic testing of the Iden3 circuits, using automated and manual processes. We reviewed only the circuits and documentation, not any on-chain or off-chain verifiers or provers.

Observations and Impact

We discovered two underconstraint issues (TOB-IDEN3-1 and TOB-IDEN3-2) in the circuits, the first of which allows bypassing revocation and expiration and could have been caught using Circomspect. Otherwise, the circuits are clear and complexity is managed well.

The public documentation that Polygon Labs provided is useful but does not contain detailed instructions for verifiers to safely integrate new features such as linked queries. Much of the overall system's security will depend on the verifiers doing suitable out-of-circuit checks and managing the overall state. These components were not included in the present review.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Iden3 take the following steps :

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Choose a standard, variable-length sponge hash construction.** The current system uses Poseidon permutations of various lengths. We recommend using a single 3-wide or 5-wide permutation inside a variable-length sponge hash instead.
- **Improve tooling and documentation around validator responsibilities.** Validators must verify many different aspects of query proofs, including timestamps, issuer revocation state, nullifiers, link IDs, challenges, and on-chain identities. We recommend producing both detailed, step-by-step validation procedure documents and tooling/SDK support for validators.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	Count
High	1
Medium	0
Low	0
Informational	5
Undetermined	1

CATEGORY BREAKDOWN

Category	Count
Cryptography	7

Project Goals

The engagement was scoped to provide a security assessment of the Iden3 Circom circuits. Specifically, we sought to answer the following non-exhaustive list of questions:

- Can a malicious prover generate a proof for credentials that they do not own?
- Can a malicious prover generate incorrect query results for credentials?
- Can issuers safely revoke credentials?
- Do the circuits enforce authorization for credential use?
- Do the circuits preserve privacy for users?
- Can users generate multiple nullifiers for a single credential?

Project Targets

The engagement involved a review and testing of the following target.

Iden3 circuits

Repository	https://github.com/iden3/circuits
Version	`7a1e04de3e5f3a9f0cfb27a43c9f41c986c1b9ed`
Type	Circom circuits
Platform	Native, EVM

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- Manual review of documentation and circuit code
- Automated review using Circomspect and internal tooling

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- We did not review any on-chain or off-chain verifiers or provers. We recommend a separate audit for the smart contracts and off-chain verifier libraries.
- We did not review the Reverse Hash Service or any specific ZK proof flows.
- We did not review any Groth16 trusted setup procedures or results.

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Unsafe use of Num2Bits in multiple circuits	Cryptography	High
2	EdDSA R value is not constrained to be on-curve	Cryptography	Undetermined
3	Lack of domain separation in hash functions	Cryptography	Informational
4	SpongeHash is not a sponge hash	Cryptography	Informational
5	Ambiguous padding in SpongeHash	Cryptography	Informational
6	Signature challenge does not bind claim or query	Cryptography	Informational
7	Linked queries can prove expired or revoked claims	Cryptography	Informational

Detailed Findings

1. Unsafe use of Num2Bits in multiple circuits	
Severity: **High**	Difficulty: **Low**
Type: Cryptography	Finding ID: TOB-IDEN3-1
Target: `Multiple (see below)`	

Description

Multiple circuits call `Num2Bits(254)` and `Num2Bits(256)` when working with field elements of the BN-254 prime field. These templates do not enforce uniqueness of the bit decompositions, allowing malicious provers to bypass token expiration or revocation.

The CircomLib `Num2Bits(n)` template allows the prover to generate the `out` array and then ensures that each element of the array is zero or one and that the following equation holds:

$$\sum_{i=0}^{n-1} out[i] \cdot 2^{n-i-1} = in$$

When n is greater than 253, the left-hand side of the verification equation can wrap around the field modulus. Thus a field element can be decomposed into multiple bit strings—for example, the bit string corresponding to either x or x + p (where p is the prime of the field) for 254-bit values, or x + k * p (for small values of k) for 256-bit values.

Num2Bits(254)

1. `getClaimHeader`: `circuits/lib/utils/claimUtils.circom#L92`
   This template could allow an attacker to produce a different set of `claimFlags` elements than expected.

2. `getClaimRevNonce`: `circuits/lib/utils/claimUtils.circom#L123`
   This template could allow an attacker to bypass revocation for a credential by having a mismatched revocation nonce. Specifically, a malicious prover could cause the `v0Bits` signal array to be the bit decomposition of `claim[4] + p`, rather than just `claim[4]`. Then the output of the `getClaimRevNonce` template would be a value that is not in the revocation tree.

3. `getClaimExpiration`: circuits/lib/utils/claimUtils.circom#L314
   This template could allow an attacker to bypass expiration if the bit decomposition produces a 64-bit integer representing a time far in the future.

4. `TakeNBits`: circuits/lib/utils/idUtils.circom#L106
   This template could allow an attacker to malign the genesis signal in the `ProfileID` circuit to produce an incorrect output. Any more significant impact is currently unknown.

5. `LessThan254`: circuits/lib/query/comparators.circom#L62-L66
   This template could lead to incorrect query results for `LessThan`/`GreaterThan` queries.

Num2Bits(256)

1. `cutId`: circuits/lib/utils/idUtils.circom#L181
   In the authentication path, the output of this template is used to verify inclusion in a sparse Merkle tree, so an incorrect value will produce a failure at that step.

2. `cutState`: circuits/lib/utils/idUtils.circom#L194
   The same outcome will occur with the `cutState` template as with `cutId`.

Exploit Scenario
Alice has a credential attesting to her identity. Mallory steals Alice's device. Alice issues a revocation for her auth credential. Mallory modifies the witness generation to produce the wrong revocation nonce. Mallory uses the unrevoked credential to perform an action impersonating Alice.

Recommendations
Short term, update the circuits to use `Num2Bits_strict()` instead of `Num2Bits(254)` or `Num2Bits(256)`.

Long term, use Circomspect to detect these issues before they are committed into code.

2. EdDSA R value is not constrained to be on-curve	
Severity: **Undetermined**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-IDEN3-2
Target: `circuits/lib/utils/claimUtils.circom`	

Description

The CircomLib `EdDSAPoseidonVerifier` template accepts the EdDSA prover commitment R value as a coordinate pair `(R8X, R8Y)` but does not check that these coordinates lie on the Baby JubJub curve. Polygon Labs' `verifyClaimSignature` template passes these values directly from the prover to the `EdDSAPoseidonVerifier` template.

While we did not identify a concrete attack exploiting this missing check, unexpected behavior can result from passing off-curve points. For example, supplying the point `(0, 0)` as the R value causes the right-hand side of the verification equation to always also be `(0, 0)`, breaking the general assumption that elliptic curve point addition is injective.

Failing to enforce the on-curve check could potentially enable more intricate cryptographic attacks leading to signature forgeries and thus unauthorized use of credentials.

Exploit Scenario

An attacker discovers a signature forgery attack exploiting off-curve R values. They use this ability to authenticate themselves to an on-chain service with another user's credentials, enabling them to steal funds from that user.

Recommendations

Short term, use the CircomLib `BabyCheck` circuit to validate prover-supplied elliptic curve points.

Long term, use Circom signal tags to enforce preconditions such as on-curve checks and range checks.

3. Lack of domain separation in hash functions	
Severity: **Informational**	Difficulty: **Medium**
Type: Cryptography	Finding ID: TOB-IDEN3-3
Target: `circuits/`	

Description

The Iden3 circuits do not include domain separation in any calls to Poseidon, potentially allowing for collisions between hashes intended for different purposes. While we did not identify any concrete attacks due to the lack of domain separation, several attacks are prevented because the relevant hash instantiations use different input widths. Including domain separation does not increase the number of circuit constraints but makes analysis of the protocol much simpler.

As an example of a potential hash collision, the Iden3 circuits distinguish between genesis IDs and profile IDs by checking whether the prover-provided state is equal to the corresponding bits of the user ID.

```
signal isStateGenesis <== IsEqual()([cutId, cutState]);
```

Figure 3.1: Check distinguishing genesis IDs from profile IDs (`circuits/circuits/auth/authV2.circom#144`)

If a prover could find a valid profile ID that was also a valid identity state, then they could equivocate on whether the supplied user ID was in fact a genesis ID or a profile ID.

Identity states are equal to the Poseidon hash of three prover-provided field elements, while profile IDs are the Poseidon hash of two prover-provided field elements. This prevents any attack in the current protocol, but future changes to the protocol could easily cause the lengths of these hashes to coincide. The lack of collisions between different hash function use cases should not depend on a contingent property such as input length but should instead be guaranteed by domain separation prefixes.

Exploit Scenario

Future changes to the protocol remove the `RootsTreeRoot` component of the identity state hash. An attacker creates an identity that can be interpreted as both a genesis identity and a profile identity. They use this ability to break protocol invariants, such as by generating two different nullifiers from the same credential.

Recommendations

Short term, replace instantiations of `Poseidon(t)` with instantiations of `PoseidonEx(t, 1)`. Use the `initialState` input to supply a domain separation value that is unique to each hash instance.

Long term, consider using a unified domain separation framework such as the SAFE API.

4. SpongeHash is not a sponge hash	
Severity: **Informational**	Difficulty: **Not Applicable**
Type: Cryptography	Finding ID: TOB-IDEN3-4
Target: `circuits/lib/utils/spongeHash.circom`	

Description

The SpongeHash template implements a variable-length hash function on top of the Poseidon permutation. However, the specific hash function implemented is a Merkle–Damgård construction rather than a sponge construction.

Sponge constructions are distinguished by their use of an internal state called "capacity" that is never directly overwritten nor output. In contrast, Merkle–Damgård constructions output their intermediate state directly as the final result.

While the typical attacks against Merkle–Damgård constructions do not affect the current Iden3 protocol, we recommend adopting a true sponge construction, as recommended by the Poseidon paper. Sponge constructions are easier to analyze and satisfy stronger cryptographic security definitions than Merkle–Damgård constructions.

Recommendations

Short term, use the `PoseidonEx` template to create a sponge construction, feeding the first output of each step into the `initialState` input of the next. Return the second output value, rather than the first, as the final result. For an example implemented by a previous client at our recommendation, see `PoseidonSponge` from the Succinct Labs Telepathy project. Ensure that padding is unambiguous when adapting for variable-length use cases.

Long term, use a consistent domain separation scheme such as those described in section 4.2 of the Poseidon paper or the SAFE API.

5. Ambiguous padding in SpongeHash

Severity: **Informational**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-IDEN3-5
Target: `circuits/lib/utils/spongeHash.circom`	

Description
The SpongeHash template implements a parameterizable fixed-input-length hash function on top of the Poseidon permutation. However, the hash function does not include the input or output length in a domain separation component, as recommended by the Poseidon authors.

Furthermore, the SpongeHash template instantiation determines a fixed-input-length hash function accepting inputs of `maxArrayLength` but is used for variable-length hashing in the `QueryHash` circuit. Unused field elements are assumed to be zero, leading to collisions between short input-length arrays and arrays ending in all zeroes. This does not currently cause an exploitable vulnerability because the SpongeHash result is itself immediately hashed together with the array length value.

Exploit Scenario
Future features use SpongeHash for variable-length arrays and fail to include the array length value in the final hash, leading to a security vulnerability.

Recommendations
Short term, initialize the sponge capacity to indicate the hash purpose and the input length.

Long term, use a consistent domain separation scheme such as those described in section 4.2 of the Poseidon paper or the SAFE API.

6. Signature challenge does not bind claim or query

Severity: **Informational**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-IDEN3-6
Target: `circuits/lib/auth/authV2.circom`	

Description
For a user with a Baby JubJub key to prove a query result on a claim issued to them, the user must supply a signature over a challenge as part of the private circuit inputs. For off-chain verification, the challenge is generated randomly by the verifier, while for on-chain verification, the challenge is equal to the `msg.sender` address of the user sending the proof.

Because the signature challenge does not include data about the specific claim or query being revealed, a signature could be reused in a context that the signer did not intend. This is mitigated because the signature itself is a private input to the circuit and thus never revealed. However, future use cases, such as hardware wallets or delegated provers, may wish to enforce a trust boundary between the private signing key and the prover of the ZK circuit.

Exploit Scenario
Alice has a credential that allows her to access an on-chain vault containing a large amount of ETH. To protect against attackers who may be able to compromise her computer, Alice uses a separate offline hardware wallet to authorize specific query proofs and does not store the private key on her internet-connected laptop.

Mallory, an attacker, compromises Alice's online computer and observes the private inputs to her ZK prover. She thus learns the contents of Alice's claims and observes a signature over Alice's ETH `msg.sender` value.

Mallory uses the observed values to prove a query authorizing a large transfer out of Alice's vault, stealing her funds.

Recommendations
Short term, document that the signature input must be kept secret to prevent unauthorized use.

Long term, hash the public input challenge alongside data (e.g., the claim hash, query hash, and link nonce) binding the challenge to a specific use.

7. Linked queries can prove expired or revoked claims

Severity: **Informational**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-IDEN3-7
Target: `circuits/lib/linked/multiQuery.circom`	

Description
To efficiently support multiple queries over a single claim, Iden3 allows linked queries. The `credentialAtomicQueryV3` circuit verifies that a specific claim was issued to a prover and then outputs a public `linkID` value that is bound to a secret `linkNonce` value chosen by the prover, who can then use that `linkNonce` with the `LinkedMultiQuery` circuit to prove up to 10 queries over the single authenticated claim.

Because credential authentication, including expiration and revocation checks, is not revalidated in `LinkedMultiQuery`, a malicious user could generate a `linkID` and cache it even after the credential expires or is revoked. While the intended use of linked queries is that the `linkID` is consumed and invalidated within a short time after generation, we did not see any documentation warning verifiers to invalidate `linkID`s after any specific time.

Exploit Scenario
Bob takes out a loan of $1M and Alice issues a credential to Bob attesting that he has $1M in a bank account.

Bob generates many `linkID`s with several verifiers using the credential. Bob then withdraws his funds from the account and repays the loan. Alice revokes the credential.

Bob continues using the `linkID`s that he previously generated to prove that he has over $1M in assets, despite not actually possessing the funds.

Recommendations
Short term, document recommendations for `linkID` validity periods.

Long term, include the issuer revocation state and claim expiry time in the `linkID` hash. Accept a timestamp as input to `LinkedMultiQuery` and validate non-expiration. Output the issuer revocation state to allow the verifier to check that it is recent.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Quality Findings

These findings are not security issues but may present opportunities for improved code clarity or efficiency.

- The `oldNewNotEqual` signal is nonzero when the `oldUserState` and `newUserstate` signals *are* equal. Consider removing the `Not` from the name.

```
signal oldNewNotEqual <== IsEqual()([oldUserState, newUserState]);
```

Figure B.1: Confusingly named `oldNewNotEqual` signal (`circuits/circuits/lib/stateTransition.circom#58`)

- The `SafeOne` template inhibits compiler optimizations unnecessarily. While the Circom optimizer does remove linear constraints on private inputs, it preserves the overall publicly visible behavior of the circuit. Because verifiers rely only on the public inputs and outputs, changes to constraints among private variables are not a security issue as long as knowledge soundness is preserved.

- The `TakeNBits` template does not restrict the `n` parameter to be less than 254 bits. There are no problematic instantiations currently, but an assertion should be added as a defensive measure.

```
// Take least significant n bits
template TakeNBits(n) {
```

Figure B.2: `TakeNBits` functions with `n` less than or equal to 254 bits (`circuits/circuits/lib/utils/idUtils.circom#101–102`)

- The `idOwnershipLevels` input's comment line is duplicated.

```
/*
…
idOwnershipLevels - Merkle tree depth level for personal claims
issuerLevels - Merkle tree depth level for claims issued by the issuer
claimLevels - Merkle tree depth level for claim JSON-LD document
maxValueArraySize - Number of elements in comparison array for in/notin
operation if level = 3 number of values for
comparison ["1", "2", "3"]
idOwnershipLevels - Merkle tree depth level for personal claims
onChainLevels - Merkle tree depth level for Auth claim on-chain
*/
```

Figure B.3: Duplicated comment lines (`circuits/circuits/onchain/credentialAtomicQueryV3OnChain.circom#27–33`)

- The `InWithDynamicArraySize` template instantiates a `LessThan(9)` component for each element of the `value` array. This could be made more efficient by instead using a mask-generating template such as the following.

```
// Generate an array consisting of size elements of value 1
// followed by (maxSize - size) elements of value 0
// Template is unsatisfiable when size is zero
template MaskArray(maxSize) {
    signal input size;
    signal output mask[maxSize];

    var sum = 0;
    for (var i=0; i<maxSize; i++) {
        mask[i] <-- (i < size);
        mask[0] === 1; // Inductive base case - all masks will be 0 or 1
        if (i > 0) {
            // mask[i] is either zero or equal to mask[i-1]
            mask[i]*(mask[i] - mask[i-1]) === 0;
        }
        sum += mask[i];
    }
    // Unsatisfiable when size is zero
    sum === size;
}
```

Figure B.4: Example of a mask-generating function using only `maxSize` nonlinear constraints

Additionally, rather than computing a running `OR` operation of the `isEq` array of signals, it would be more efficient to simply sum the elements and check whether the sum is zero at the end.

- The comment for the `verifyExpirationTime` template is backwards: the `timestamp` value is required to be less than the expiration time.

```
// verify that the claim has expiration time and it is less then timestamp
template verifyExpirationTime() {
```

Figure B.5: Incorrect comment describing `verifyExpirationTime` (`circuits/circuits/lib/utils/claimUtils.circom#284–285`)

C. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On April 19, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the Iden3 team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the seven issues described in this report, Iden3 has resolved two issues and has not resolved the remaining five issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Unsafe use of Num2Bits in multiple circuits	Resolved
2	EdDSA R value is not constrained to be on-curve	Resolved
3	Lack of domain separation in hash functions	Unresolved
4	SpongeHash is not a sponge hash	Unresolved
5	Ambiguous padding in SpongeHash	Unresolved
6	Signature challenge does not bind claim or query	Unresolved
7	Linked queries can prove expired or revoked claims	Unresolved

Detailed Fix Review Results

TOB-IDEN3-1: Unsafe use of Num2Bits in multiple circuits
Resolved in PR #131. All uses of `Num2Bits` with sizes greater than 253 have been converted to `Num2Bits_strict`. There remain two unguarded parameterized uses of `Num2Bits` in the `AddMaxbitTag` and `AddMaxbitArrayTag` templates; however, these templates are currently unused. We recommend adding assertions to them requiring the input parameter to be strictly less than 254.

TOB-IDEN3-2: EdDSA R value is not constrained to be on-curve
Resolved in PR #131. Polygon Labs now checks that prover-supplied elliptic curve coordinates are on the correct curve via the `ForceBabyCheckIfEnabled` circuit.

TOB-IDEN3-3: Lack of domain separation in hash functions
Unresolved. Polygon Labs indicated that it may add domain separation in future versions but that this would break existing credentials. Polygon Labs provided the following reasoning:

> *Since fixing this issue would introduce breaking changes to the whole protocol (e.g., generated identifiers, hashes in Merkle trees, claim signatures, etc., will be different) and is not critical, we will postpone implementation of suggested changes to the next major protocol update.*

TOB-IDEN3-4: SpongeHash is not a sponge hash
Unresolved. Polygon Labs indicated that it may modify the hash function in future versions but that this would break existing credentials. For issues TOB-IDEN3-4 and TOB-IDEN3-5, Polygon Labs provided the following reasoning:

> *Fixing these issues would introduce breaking changes to some components of the protocol (e.g., used in credential Merklization algorithm), so we will implement them in the next major versions of those components.*

TOB-IDEN3-5: Ambiguous padding in SpongeHash
Unresolved. See the quote provided for TOB-IDEN3-4 above.

TOB-IDEN3-6: Signature challenge does not bind claim or query
Unresolved. Polygon Labs provided the following reasoning:

> *Relatively small change, but touches many components; will be implemented in the next version of circuits or addressed on a higher level (requirement on how challenge is generated by the verifier/user).*

TOB-IDEN3-7: Linked queries can prove expired or revoked claims
Unresolved. Polygon Labs provided the following reasoning:

Linked Queries are totally dependent on base Query proof for revocation and expiration checks and are not valid without them. Our off-chain verification libraries accept linked proofs only if they are sent in one proof response batch (in one JWZ container). For on-chain verification (since for now proofs can only be sent in separate transactions), dApps can rely on block timestamp/number of base query proof tx if they need to get a point in time on which base query together with all linked to it query proofs were valid. These things will be documented properly soon.

D. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.