INVERSED TECH

Audit Report

Iron Fish Protocol & ZKP Circuits

Daniel Benarroch, Bryan Gillespie, and Aurélien Nicolas

June 1st, 2023

Contents

1 Overview

We present the findings of our review of the Iron Fish Private Multi-Asset Transfer protocol (from now on "the protocol" or "the IF protocol"). The protocol, based on the Sapling protocol built and deployed in the Zcash blockchain, was extended to support multiple asset types, including new functionalities for the issuance and burning of user-specified assets. The protocol uses Zero-Knowledge Proof technology to hide sensitive information of a transaction, creating confidentiality around the type of asset and the amount being transferred, and enabling anonymity of the sender and receiver of a transfer. It is important to note that the issuance and burning mechanisms are transparent.

There are a few important changes that we kept in mind during our review:

- Any given wallet (defined by the private key) can only generate a single public address (compared to Sapling where a diversifier is used to generate multiple addresses from the same private key).
- The Sender address is included in the transaction for traceability properties, as someone who receives funds can now identify the origin.

During the review, we focused on both the cryptographic and engineering aspects of the protocol. We reviewed the design of the protocol, the Rust implementation of the different components, as well as the ZKP circuit implementations.

The team in charge of the audit was composed of Daniel Benarroch, Bryan Gillespie, and Aurélien Nicolas.

1.1 Resources

The audit was done on two specific commits of the Iron Fish code base:

- The initial reviews were done on the tagged version v0.1.62 on the `master` branch, specifically
  - Commit hash 4ab43f39ed7d8692dc82efe30f12fe2fad5b89b7
- After some of the issues found were fixed, the rest of the review was done on the tagged version v0.1.76, which was the latest tagged version before the mainnet release.
- For the review of the fix of the **Asset ID Malleability Vulnerability**, described in Section 2, the review was done on the `multiasset-changes` branch, specifically
  - Commit hash a31cd7194efd972fdcc82738343b82590446356b

We reviewed the following code components:

- The `ironfish-rust` folder where the cryptographic protocol is implemented

  - the keys structure implementation in `keys`
  - the transaction structure implementation in `transaction`
  - the note, note commitment and Merkle tree implementations
- The `ironfish-zkp` folder where the ZKP circuits are implemented
  - the *mint* circuit, which proves that an issuer minted a new asset properly
  - the *spend* circuit, which proves that an existing note was spent correctly by the sender
  - the *output* circuit, which proves that the new note created for the receiver was generated correctly by the sender
  - the *value commitment* functionality, where a homomorphic commitment is used to ensure the balance of the transaction
- We also reviewed other components as part of the extended protocol and the consensus validation, to ensure protocol soundness. This also included the node and client validation in TypeScript and some of the Node.js components.

In addition, the following resources were used:

- The Zcash Protocol Specification (Hopwood et al. 2016) was used as an important reference for verifying correct implementation of components of the protocol based on the Sapling protocol
- The Iron Fish Whitepaper (Iron Fish Team 2020) was used to get a high level view of the protocol.

1.2 Summary of Findings

We categorize our review in four type of issues

- **Critical Issues**, denoted by the **color red**, describe findings that have a practical vulnerability in the blockchain protocol, for which we have implemented tests showing the severity of the finding.
- **Warning Issues**, denoted by the **color yellow**, describe findings that do not have a practical vulnerability because of some external reason to the context of the function. These issues should be fixed to prevent future vulnerabilities to arise.
- **Informational Issues**, denoted by the **color blue**, describe findings that do not carry a risk of vulnerability, but that we believe do not follow best practices. All informational issues do not have any practical risk.
- **No Issue Found**, denoted by the **color green**, describes the components of the protocol that we reviewed and for which we found no issues.

We summarize the most important issues (a full list of findings can be found in Section 3). It is important to note that as of the writing of this report, **all issues found in the audit have been fixed and no further issues were found**.

1.2.1 Critical Issues

1. **Section 2 — Asset ID malleability vulnerability (Fixed)**

   This vulnerability was discovered as a result of two critical findings in the code:

   - **INV-1 — Hash function for value commitment generators is not pseudo-random (Fixed)**

     First, the hash function used to generate the Asset type generator was not pseudorandom. This issue has been solved by replacing the Pedersen hash with a PRF.

   - **INV-2 — Transaction proofs don't verify knowledge of generator hash preimage (Fixed)**

     Second, there was no verification of the asset type in the output proof. This issue has been solved by adding an asset type derivation verification in the Output circuit.

   These two issues were used in the vulnerability discovered that allows an attacker to issue any type of asset in full confidentiality. This issue is fully described in Section 2.

   Fixed. This finding was first reported to the Iron Fish team on Tuesday, March 7th. As of version `v0.1.76`, this issue has been fixed.

2. **INV-24 — Pure mint transactions are arbitrarily replayable (Fixed)**

   A transaction which mints a custom asset, spends no notes, and specifies zero network fee is allowed by node functioning and network consensus rules, and may be replayed on the blockchain multiple times. Without spend notes, there is no consensus mechanism to detect a replay of the transaction.

   Fixed. This issue was first reported to the Iron Fish team on Thursday, May 4th. As of version `v1.3.0`, this issue has been fixed.

1.2.2 Warning Issues

1. **INV-25 — Inline double-spend transactions allow memory pool contamination (Fixed)**

   Transactions which attempt to spend the same note multiple times in a single transaction are accepted into a node's mempool when received from a peer node, and are incorporated into mining block templates from a node's mempool.

   Fixed. This issue was first reported to the Iron Fish team on Thursday, May 4th. As of version `v1.2.0`, this issue has been fixed.

1.2.3 Informational Issues

We remind the reader that informational issues do not carry any practical risk to the protocol.

1. **INV-3 — Value commitments have at most 126 bits of security**
2. **INV-4 — Subtle design of asset IDs could be a development hazard**
3. **INV-5 — Unnecessary mint circuit hash function**
4. **INV-13 — Existing errors in the documentation of the code in note.rs**
5. **INV-20 — Unit tests do not verify key derivation correctness**
6. **INV-21 — Key derivation algorithm diverges from specification**
7. **INV-22 — Keyspace overlap with Zcash Sapling shielded pool**
8. **INV-26 — Insufficient validation of mempool transactions**
9. **INV-27 — No validation of transactions received from the API**
10. **INV-28 — Confusing naming in Spend.commitment attribute**
11. **INV-30 — Unreliable fixtures in unit tests**
12. **INV-31 — Rust code implemented in Node-API wrapper is untested**
13. **INV-32 — Iron Fish whitepaper diverges from implementation**
14. **INV-33 — Iron Fish protocol lacks a formal security specification**
15. **INV-34 — Missing framework requirements from Zcash imports**
16. **INV-35 — Client `addTransaction` API has unintuitive behavior**

1.3 Unit Tests

During the review, various unit tests were implemented to verify the behavior of code and confirm the presence of vulnerabilities. The following new unit tests are made available with this report, under the `Inversed-Tech/ironfish-audit` fork of the Iron Fish repository. Tests are based on the `v0.1.76` release of the Iron Fish client unless otherwise specified.

Branch `audit-tests-base-v0.1.76` contains the majority of tests which do not require changes to library code or depend on an older version of the Iron Fish client.

- **TEST-1**: Verify that network nodes will accept “inline double-spend” transactions from the network, and will forward such transactions to peers

  Source: `ironfish/src/network/peerNetwork.test.ts:1063-1152`

- **TEST-2**: Verify that nodes will incorporate “inline double-spend” transactions contained in the mempool into miner block templates

  Source: `ironfish/src/mining/manager.test.ts:96-128`

- **TEST-3**: Verify that nodes throw an error when attempting to submit a block template containing an “inline double-spend” transaction to a miner

  Source: `ironfish/src/mining/manager.test.ts:130-171`

- **TEST-4**: Verify network nodes will accept “pure mint” transactions with zero fee from the network, and will forward such transactions to peers

  Source: `ironfish/src/network/peerNetwork.test.ts:1154-1260`

- **TEST-5**: Verify that nodes will incorporate “pure mint” transactions with zero fee contained in the mempool into miner block templates, and will submit such block templates to miners

  Source: `ironfish/src/mining/manager.test.ts:173-219`

- **TEST-6**: Verify that the blockchain will accept multiple instances of the same “pure mint” transaction with zero fee

  Source: `ironfish/src/blockchain/blockchain.test.ts:806-855`

- **TEST-7**: Verify that all individual steps of the key generation procedure produce correct results

  Source: `ironfish-rust/src/keys/test.rs:36-151`

Branch `asset-malleability-test` contains tests verifying the asset ID malleability vulnerability on the audit target release, `v0.1.62`.

- **TEST-8**: Validate the arbitrary mint exploit described in Section 2.2

  `ironfish-rust/src/transaction/mod.rs:824-912`

Branch `negative-transaction-fee-test` contains a test which required modifications to underlying library code to implement.

- **TEST-9**: Verify that nodes correctly reject transactions from the network which specify a negative fee, and do not forward such transactions to peers

  `ironfish/src/network/peerNetwork.test.ts:1058-1134`

2 Asset ID Malleability Vulnerability

In March 2023, Inversed auditors discovered a critical vulnerability in the Iron Fish protocol: the procedure used to create and validate identifiers for custom assets had a malleability issue allowing an attacker to break the invariant that "the total input and output values of a transaction must be balanced for each asset type". Auditors were further able to identify a practical exploit on the audit target commit `v0.1.62` which leverages this vulnerability to secretly mint arbitrary assets on chain, including the Iron Fish native asset. **However, the mitigations that have been implemented since the vulnerability was discovered have resolved the issue, making this exploit incompatible with the newest client releases**.

A working unit test demonstrating the exploit on release `v0.1.62` was developed during the review. See TEST-8 in the index of unit tests for a static repository link for this code.

In the remainder of this section, we review the underlying vulnerability discovered in the Iron Fish protocol, provide a detailed description of the arbitrary minting exploit, and discuss how the currently implemented mitigations invalidate the exploit on recent client releases.

2.1 Vulnerability Description

The Iron Fish protocol represents generic assets on chain by hashing the fields of an asset specification to produce an elliptic curve group generator used by transaction *value commitments*, which are subsequently used to verify that the input and output values included in a transaction are balanced. Iron Fish value commitments are implemented using the Pedersen commitment scheme, whose security properties rely on the assumption that no discrete log relations are known between the group generators to which the committed values are attached.

As an example, suppose that two generators G and H in a cryptographic group are used to produce Pedersen commitments for vectors of length two. Then the commitment produced for the pair of values $(10, 20)$ would be the group element

$$\texttt{commitment}(10, 20) = 10 * G + 20 * H$$

Commitment schemes should have the *binding* property, meaning that it is computationally infeasible to find two distinct vectors producing the same commitment; however, if an algebraic relation is known between generators, then this property fails. For instance, in the previous example, if we know that G and H satisfy the relation $H = 2 * G$, then we can easily find another vector producing an identical commitment, such as

$$\texttt{commitment}(30, 10) = 30 * G + 10 * H = 10 * G + 20 * H$$

In general, it is computationally infeasible to find discrete log relations like this between group generators sampled uniformly at random from a cryptographic group, so generators selected by a pseudo-random function are typically suitable for producing secure Pedersen commitments. Two potential issues related to this point were observed in the Iron Fish `v0.1.62` implementation.

First (finding INV-1), the Sapling `PedersenHash` primitive used to produce asset generators is not pseudo-random. The Zcash specification (Version 2022.3.8, p. 78) states that `PedersenHash` is "required to be collision-resistant between inputs of fixed length, for a given personalization input", but that "no other security properties commonly associated with hash functions are needed." In particular, Pedersen hashing is *malleable*, meaning that similar inputs produce hash values that are themselves similar in a predictable way.

Second (finding INV-2), the asset generator included in a transaction Output was not checked to be the hash image of a known asset specification, allowing an attacker to select an arbitrary asset generator to be incorporated into the transaction's aggregated value commitment. Notably, this bypasses the chosen hash-to-group function entirely, and breaks the assumption that only independent (random) generators are used in the underlying Pedersen commitment. In the following, we give a demonstration of how this second issue leads to a significant violation of Iron Fish's protocol integrity.

2.2 Unrestricted Minting Exploit

On Iron Fish release `v0.1.62`, a valid transaction can be constructed which has two related Output components, but no Spend, Mint, or Burn components. The Outputs of the transaction are designed so that their resulting value commitments cancel each other out in the binding signature phase of transaction creation, thereby satisfying the protocol which confirms balance between the input and output values. The following pseudocode describes the procedure:

1. Create an empty Transaction `tx`.
2. Create an arbitrary Output component `out1` with some value `val`, and add it to `tx`.
3. Create another Output component `out2` identical to `out1`, except that the asset generator of `out2` is `-1` times the asset generator of `out1`. Add `out2` to `tx`.
4. If the asset specified by either of the outputs is the native Iron Fish asset, additionally subtract `val` from the native asset's total in the table of value balances for `tx`.
5. Sign and post `tx` to the network as usual.

Recall that when validating the balance of input and output values in Iron Fish transactions, the value commitments of the transaction components are summed with appropriate signs, and the subsequent binding signature stage cannot be executed unless the transaction values balance in this sum. In the

above, the value commitments associated with the two Output components are

$$\texttt{value\_commitment}(\texttt{out1}) = \texttt{val} * G + \texttt{rand1} * G_{rcv}$$

and

$$\texttt{value\_commitment}(\texttt{out2}) = \texttt{val} * (-G) + \texttt{rand2} * G_{rcv}$$

where G is the asset generator of `out1`, and G_{rcv} is a fixed generator used to represent the random blinding component of all Iron Fish value commitments, independent of asset type. In particular, the G component vanishes when summing these commitments, resulting in a total value commitment of $(\texttt{rand1} + \texttt{rand2}) * G_{rcv}$ which allows the exploit transaction to pass the value balance verification step.

The attack thus produces a Transaction which is accepted by the client software validation protocol, and produces two notes associated with the Outputs `out1` and `out2`. The note from `out1` is of arbitrary asset type and arbitrary value, which can be used on the network as usual, while the note from `out2` with negative asset generator is not likely to correspond with an asset type of interest, but can safely be ignored. Since the number of components of various types contained in a transaction is publicly visible, one or more dummy Spend components with value 0 may be added to `tx` so that the transaction metadata looks normal to network participants.

2.3 Analysis of Mitigations

The asset ID malleability vulnerability present in release `v0.1.62` appears to have been successfully addressed by mitigations implemented in release `v0.1.76`. The idea of the mitigations is to require that asset generators be derived using a pseudo-random function from a known preimage, ensuring that users are unable to exert influence over the choice of asset generator except by random sampling from a space where adversarial generators are very sparse.

The mitigations are implemented in two main parts. The first, which is most relevant to the exploit described in this report, is a new check that was added to the zk-SNARK proof included with transaction Outputs, which verifies that the prover knows a hash preimage of the specified asset generator. This ensures that a user's ability to specify an Output asset type with an adversarial generator is related to the preimage resistance of the hash function used to produce asset generators by the protocol. This single check is enough to verify that a hash preimage is known for any asset specified by an Iron Fish transaction component, by the following reasoning: (1) an Output component checks for knowledge of a preimage explicitly; (2) a Spend component references a Note created by an Output component; and (3) Mint and Burn components give an asset preimage explicitly.

The second part of the mitigations is a change in the hash function used to produce an asset's generator from its name, metadata, and public minting address. In the most recent releases of Iron Fish, the

Sapling `PedersenHash` primitive has been replaced by the `Blake2s` cryptographic hash function, which is pseudo-random, and in particular has the strong preimage resistance needed to ensure that users are unable to find preimages of adversarial asset generators.

The specific implementation used to produce an asset's generator from its name, metadata, and public minting address is an adaptation of Zcash Sapling's "group hash into Jubjub" (Hopwood et al. 2016, sec. 5.4.9.5), which uses a rejection-sampling approach to finding a `Blake2s` hash which can be directly deserialized to a Jubjub elliptic curve point. This necessitates an additional field for Iron Fish assets, an 8-bit unsigned integer "nonce" which is chosen to ensure that a hash output can be interpreted as a serialized Jubjub point for use as an asset generator. We remark on a few technical considerations surrounding the adapted group hash algorithm:

- Since the nonce field is part of the data used to derive an asset's generator, this field must be considered as a first-class member of an asset specification. That is, two asset specifications which differ only in their nonce value must be interpreted by protocol functionality as different assets. The current implementation handles this detail correctly.
- The hash-to-group function works in two steps: first, an "asset ID" field is produced as the `Blake2s` hash output of an asset's name, metadata, public minting address, and nonce. Then the `Blake2s` hash output of this asset ID field is interpreted directly as the serialized form of a Jubjub elliptic curve point. This design is advantageous because it decouples the specific fields of an asset specification from the zk-SNARK proof demonstrating knowledge of the hash preimage of an asset generator, and it also makes this zk-SNARK proof more efficient because the hash preimage of an asset generator is a small, fixed size of 32 bytes.
- The asset generator produced by the group hash algorithm is a point on the Jubjub elliptic curve which is not guaranteed to lie in the prime order subgroup of points suitable for use in Pedersen commitments. This means that two elliptic curve points are associated with an asset, the "asset generator" produced by deserializing a hashed asset ID, and the "value commitment generator" which is a related point in the prime order elliptic curve group obtained by "clearing the cofactor" of the asset generator. The use of both of these subtly different points in various parts of the codebase is a potential anti-pattern and should be carefully planned around. See finding INV-4 for recommendations.
- It is important to check that the asset generator produced by this approach is not a point of low order on the Jubjub elliptic curve group, as such a point does not produce a suitable value commitment generator for Pedersen value commitments. The current implementation verifies this property when constructing assets.
- It is advisable to use customized personalization values or prefix bytes for the applications of `Blake2s` involved in producing Iron Fish asset generators to proactively prevent collisions with

hashes produced by other applications. The current implementation correctly makes use of customized personalization values.

The concrete security level of the revised scheme for generic asset IDs against an unrestricted minting exploit like the one described in Section 2.2 is estimated to be 126 bits, which is suitable for production use. See finding INV-3 for more details on the analysis leading to this estimate.

3 Complete List of Findings

3.1 Generic Assets

3.1.1 INV-1 — Hash function for value commitment generators is not pseudo-random

The Pedersen hash function used to produce the value commitment generators for generic Iron Fish assets is not a pseudo-random function. This fact may allow network participants to produce assets with specially targeted generators which could break the binding property of Iron Fish value commitments.

This issue has been fixed.

Description. This vulnerability and its consequences are discussed in detail in Section 2. Effective mitigations for this vulnerability have been implemented since Iron Fish release `v0.1.76`.

Fixed. This finding was first reported to the Iron Fish team on Tuesday, March 7th. As of version `v0.1.76`, this issue has been fixed. The Iron Fish team implemented a PRF function, Blake2s, instead of the Pedersen commitment function, to generate the asset ID generators.

3.1.2 INV-2 — Transaction proofs do not verify knowledge of generator hash preimage

Transactions allow specifying an arbitrary asset generator for use in the construction of value commitments, without demonstrating knowledge of an asset specification providing a hash preimage of this generator.

This issue has been fixed.

Description. This vulnerability and its consequences are discussed in detail in Section 2. Effective mitigations for this vulnerability have been implemented since Iron Fish release `v0.1.76`.

Fixed. This finding was first reported to the Iron Fish team on Tuesday, March 7th. As of version `v0.1.76`, this issue has been fixed. The Iron Fish team implemented a Blake2s verification inside the Output circuit that ensure that the asset ID being spent in the Spend description is the same as the one being used in the Output description.

3.1.3 INV-3 — Value commitments have at most 126 bits of security

Following mitigations for the asset ID malleability vulnerability implemented in release `v0.1.76`, the concrete security of the Iron Fish value commitment scheme against malleability attacks is at most 126 bits, at which difficulty an attacker would be able to undetectably mint tokens of an arbitrary asset on the Iron Fish network.

This finding does not pose a security risk.

Description. The concrete details of the security analysis are the following. The use of the Pedersen commitment scheme for note value commitments imposes a restriction on the generators used to represent the values of different assets. If G and H are the value commitment generators representing the values of two different assets, then the committer cannot know a discrete log relation $H = kG$ between G and H, or else the scheme does not bind the committer to a single pair of values for the associated assets. In particular, if A is an asset of interest to an attacker and G is its associated generator, then in order to break binding of the value commitments representing note values on Iron Fish, it is enough for an attacker to find an adversarial asset B (mintable by the adversary, say) whose generator H satisfies a relation $uH = vG$ for some unsigned 64-bit values u and v. Upon discovery of such a generator and relation, a value commitment for asset B with value u can be used interchangeably with one for asset A with value v.

Such an asset can be discovered by the following approach: make a large hash map M whose entries associate the group element $vu^{-1}G$ with the pair (u, v) for many pairs of `u64` values u and v. (Here, u^{-1} indicates inversion in the scalar field of the Jubjub elliptic curve group.) Then generate many candidate assets B, and for each asset, check whether its associated value commitment generator H is a key in M. Once this occurs, the pair (u, v) stored with this key describes the discrete log relation between G and H, and the asset B can be used to break the binding property of the Iron Fish value commitment scheme.

The difficulty of this attack is a trade-off between the size of the hash map M and the number of candidate assets B which need to be considered. If r is the size of the Jubjub elliptic curve group, a number between 2^{251} and 2^{252}, and m is the size of the hash map M, then the expected number of candidate value commitment generators which will need to be computed and compared with M is r/m. Since constructing M requires resources (time and space) of order $O(m)$ and checking candidates requires resources (time) of order $O(r/m)$, the overall cost is minimized at around $m = \sqrt{r}$, yielding performance of around $O(\sqrt{r})$ time and space, or slightly under 126 bits of security.

A small but important point in this analysis is the following fact: it is possible to produce a hash table M of the specified form which has $m = \sqrt{r}$ entries. This is not obvious from the specification, because

two pairs (u, v) and (u', v') of 64-bit values produce the same hash table key when $v/u = v'/u'$ as rational numbers. Each fraction has a unique representation whose numerator and denominator are relatively prime (the reduced form of the fraction), so an estimate is needed for the size of the set $S = \{(u, v) \in \{1, \ldots, 2^{64} - 1\}^2 : \gcd(u, v) = 1\}$.

Such an estimate can be determined empirically by random sampling. On a trial of 20,000,000 random pairs of u64 values, 12,157,128 of them, or around 60.78%, were found to be relatively prime. This demonstrates that the size of S is at least $2^{127.2}$ with high probability, a value larger than $\sqrt{r}$ by a small constant factor. This subsequently implies that it is possible to produce a hash map M large enough to carry out the attack described above.

Recommendation. No action is necessary, as 126 bits of security is ample for modern applications. We make this note because 126 bits is somewhat lower than might be expected from the use of the 256-bit cryptographic hash function Blake2s for deriving asset generators. Such a bound on the concrete security of the Iron Fish value commitment scheme could be a relevant consideration for long-term planning of network upgrades.

3.1.4 INV-4 — Subtle design of asset IDs could be a development hazard

Following mitigations for the asset ID malleability vulnerability implemented in release `v0.1.76`, asset identifiers include two related elliptic curve points which are meant to be used in subtly different settings. This design has some potential to lead to hard-to-detect bugs or vulnerabilities during future development. The current implementation appears to use these points correctly.

This finding does not currently pose a security risk.

Description. The current scheme used by Iron Fish to represent custom user assets involves a data structure with two related elliptic curve points, an “asset generator” and a “value commitment generator”. The asset generator is an elliptic curve point on the Jubjub elliptic curve produced directly from the output of a pseudorandom function, while the value commitment generator is obtained from the asset generator by clearing cofactors, producing a point contained in the prime order Jubjub elliptic curve group. The choice to include both of these elliptic curve points in the implementation is a valid engineering decision with some practical benefits in terms of performance, but could be confusing and prone to errors during future development.

Recommendation. Add documentation describing the purpose and usage of the two elliptic curve points to help minimize confusion during future development. Consider refactoring asset identifiers to include only a single elliptic curve point.

3.2 ZKP Protocol and Circuits

3.2.1 INV-5 — Unnecessary mint circuit hash function

The hash into generator of the asset type does not have to be computed inside the `MintAsset` circuit.

This issue has been fixed.

Description. The hash of asset info is computed in the circuit `MintAsset`. However, assuming asset info is public, this is unecessary, as the info could just be included in the transaction without obfuscation. (If asset info is to become private, then there should be some blinding factor.) From there, the only purpose of the circuit `MintAsset` is to verify knowledge of `ak` and `nsk` as preimages leading to the owner's public address.

Fixed. This issue was fixed as of version `v0.1.76` by implementing the `asset_info` to `asset_id` hash computation as part of the public consensus protocol and removing this computation from the `MintAsset` circuit. In new client releases, public validation ensures the hash is properly derived.

3.2.2 INV-6 — Correct `MintAsset` circuit implementation

It was checked that the `MintAsset` circuit is properly designed and implemented with respect to the desired functionality.

No issue was found.

Description. There are two main requirements for security of the `MintAsset` circuit:

1. Ensure that each transaction signature is different to prevent linkability of issuance instances. This is satisfied, as in the `Spend` circuit, since the public key `ak`, which is witnessed in the circuit, is rerandomized with `ar`, a random seed used to produce `rk`, which is exposed publicly.
2. Ensure that there are no asset type collisions feasible across issuers. This is satisfied since the `asset_info`, which is witnessed in the circuit, includes the `owner_public_address`.
3. Ensure that `owner_public_address` is derived from the same user who owns the private key that derives the signature key used, `ak`.

3.2.3 INV-7 — Correct asset type and value commitments protocol implementation

It was verified that asset type and value commitment structs are implemented correctly.

No issue was found.

Description. We verified that the following properties were satisfied. For the asset type struct:

- Asset type must be a valid canonical elliptic curve point
- Asset type must be a cryptographically independent point, with no known relation to any other generator, and in particular, there are no collisions with `zcash_primitives::constants`
- Asset type must not be the identity element

For the value commitment struct:

- Multiple commitments for different asset types can be combined by `EC_ADD`.
- A commitment of combined types can be opened to the vector of its values per type securely, and does not allow forging values in any type. It is a homomorphic vector commitment.
- Opening requires knowledge of the pairs `(type, value)` for all types with a non-zero value.
- Opening requires knowledge of the sum of blinding factors that went into its construction.

3.2.4 INV-8 — Correct circuit implementation of value commitment functionality

It was verified that value commitments are correctly computed in the `Spend` and `Output` circuits to ensure that the correct value is used in the `bindingSignature`.

No issue was found.

Description. The following is a list of checks done to ensure that the implementation is correct.

- The asset type witnessed into the Spend and Output proofs must be connected to the components relying on it:
  - The `asset_generator` point is witnessed and passed to the value commitment logic
  - The witnessed point is passed into the note commitment as equivalent bits, including both x- and y-coordinates with congruency
- Honest users generate unpredictable value commitment randomness `rcv` using fresh entropy, and do not leak or reuse this randomness

- The note value must be in range (by construction of bits in `expose_value_commitment`)
- The `cv` witness is exposed as public input (instance) of the circuits `Spend` and `Output`; both circuits call `expose_value_commitment` once and pass the resulting bit representation of the value to subsequent logic
- The note value witness must be connected to other components relying on it
  - `note_contents` have the same bit format in Output and Spend: `asset_generator` x-coordinate then y-coordinate, then `value_bits` in little-endian order
- The function `expose_value_commitment` works correctly:
  - Allocate 64 variables, each constrained to be binary (in `AllocatedBit::alloc`), which represent the bits of a specified value commitment's value
  - Expose `cv` as a public instance variable, formatted as the (x, y) Edwards coordinates of the elliptic curve point, possibly the identity point
  - Derive `cv` from the value bits. The following relation holds:

$$\texttt{cv} = \texttt{asset\_generator} * \texttt{value} + \texttt{RCV\_BASE} * \texttt{rcv}$$

  - Return the allocated value bits at the end of the function call
  - The gadget for `EC_MUL` is used correctly (little-endian order, and no congruencies are possible with 64 bit values)
  - The gadget for `EC_ADD` is used correctly
  - The gadget for `EC_MUL_FIXED` is used correctly (with 252 bits, using 84 windows of 3-bits)
  - The blinding factor `rcv` is normally given as a field element (Jubjub scalar), but actually witnessed as bits, and congruent values are allowed as proof of knowledge of `rcv`
  - The base point used to represent `rcv` in the value commitment is a constant of the protocol, and must be a cryptographically independent point with no known relation to any other generator. This value is given by `VALUE_COMMITMENT_RANDOMNESS_GENERATOR` in module `zcash_primitives::constants`.

3.2.5 INV-9 — The Groth16 implementation prevents input malleability

It was verified that the Groth16 implementation prevents input malleability. This is accomplished by adding the constraints `input[i]*[0] == 0` for all inputs to the circuit.

No issue was found.

3.2.6 INV-10 — Correct implementation of Spend authorization signature

It was verified that the protocol correctly uses the RedJubjub signature scheme to sign the transaction hash, preventing malleability of the message, as well as properly authorizing the spending ability for the given spend notes.

No issue was found.

3.2.7 INV-11 — Correct use of signature rerandomization factor

It was verified that signature rerandomization is used correctly.

No issue was found.

Description. In order for transactions produced by the same user to be unlinkable by other network participants, the protocol uses rerandomized signature keys along with a proof of knowledge of the randomizing factor. This ensures that transaction signatures are with respect to a new keypair for each transaction, strengthening the privacy properties of the protocol.

Note. The protocol does not abide by a strong form of unlinkability due to the `SenderID` field added to the notes. More on this is discussed in INV-33.

3.2.8 INV-12 — Correct key structure validation in Spend proof

It was checked that key structure validation conducted in `Spend` proofs is correct.

No issue was found.

Description. The `Spend` proof verifies that the sender knows both `ak` and `nsk`, and that they are both derived from the same private key. Furthermore, the proof validates that these keys are related to the `owner_public_address`, which is needed to make sure the owner of the note is the one spending it.

3.3 Transactions, Notes, Merkle Trees, Nullifiers

3.3.1 INV-13 — Existing errors in the documentation of the code in note.rs

Errors were found in documentation for code in the file `note.rs` in the `ironfish-rust` library.

This finding does not pose a security risk.

Description. The following are errors in documentation in the `note.rs` code file in the `ironfish-rust` library. Specific line numbers refer to release `v0.1.76`.

- L.67 → "`along with a nullifier key that is made public`". The nullifier key is not made public, but rather, the nullifier itself is.
- L.89 → "`the spender can supply when constructing a spend`". A memo is supplied for a new note when the spender constructs an output, not a spend.
- L.198 → "`function allows the owner to decrypt`". This is incorrect, as this function requires the *spender's* info; this may have been confused with similar documentation in the function `from_owner_encrypted`.

3.3.2 INV-14 — Notes are correctly structured

It was checked that the notes used for Spends and Outputs have the correct structure.

No issue was found.

Description. The structure of the Iron Fish note implementation was confirmed to include the following components.

- The note owner as `PublicAddress`
- The value as `u64`
- The note sender as `PublicAddress`
- The asset type as `jubjub::SubgroupPoint`
- The commitment randomness as `jubjub::Fr`
- The memo field as `Memo`

These are the basic elements of any representation of funds on chain.

3.3.3 INV-15 — Correct implementation of on-chain note encryption

The encryption method for the on-chain representation of notes was checked for correctness.

No issue was found.

Description. The encrypted note is sent to the chain for two purposes: (1) in order for the receiver, or owner, of the note to decrypt its contents to be used; (2) in order for both the sender and receiver to be able to decrypt the contents of the note with the appropriate viewing keys, which may also potentially be shared with third parties.

3.3.4 INV-16 — Note commitments are correctly computed

The computation of note commitments was checked for correctness.

No issue was found.

Description. The note commitment is computed in order to be published on chain while hiding and committing to the note content. The commitment itself is added to the leaves of the public Merkle tree of note commitments, and hence must be computed as a PRF to hide properly the contents of the note. This is done in the IF protocol, using the Blake2s hash function.

3.3.5 INV-17 — Nullifiers are correctly derived

The computation of note nullifiers was checked for correctness.

No issue was found.

Description. The nullifier is the string of bytes that prevents notes to be spent more than once. It is published publicly as part of a spend description. It is important for it to not reveal anything about the note or note commitment it is nullifying, not even the location of the commitment in the tree. This is why a PRF must be used, such as Blake2s, as is used in th IF protocol.

One important aspect of the derivation is to properly construct the `rho` parameter from the note commitment and its position in the tree, which is then combined in the PRF with the nullifier key, `nk`, which is derived from the note owner private key. This is also checked in the Spend proof.

3.3.6 INV-18 — Transactions are correctly constructed from all descriptions

It was verified that posted transactions contain properly constructed parameters representing all input fields in the `ProposedTransaction` builder struct.

No issue was found.

Description. A `ProposedTransaction` builder struct is used to aggregate transaction components in the native `ironfish-rust` library prior to constructing a final signed and posted transaction for distribution to the network. It is important that the inputs of the builder struct are properly reflected in the component descriptions of the resulting posted transaction, and that all component descriptions are properly constructed.

It was verified that the following parameters for a posted transaction are properly constructed and contained in a `Transaction` object produced by the Iron Fish library code:

- Mints with `proof`, `asset`, `value`, and `authorizing_signature`
- Spends with `proof`, `value_commitment`, `root_hash`, `tree_size`, `nullifier`, and `authorizing_signature`
- Outputs with `proof` and `merkle_note`
- Burns with `asset_id` and `value`
- Transaction `fee` and `binding_signature`, witnessing the value balances and commitment randomness from other components

3.3.7 INV-19 — Correctness of Diffie-Hellman secret sharing

The Diffie-Hellman implementation used for encryption of note plaintext for sender and recipient was checked for correctness.

No issue was found.

3.4 Key Structure

3.4.1 INV-20 — Unit tests do not verify key derivation correctness

Tests implemented in the target release do not verify correctness of the individual transformations deriving key components from an initial secret key.

This finding does not currently pose a security risk.

Recommendation. Unit tests were implemented during the audit which verify the correctness of individual transformations in the key derivation algorithm. Merge these tests into the main branch of the Iron Fish repository.

Tests. Unit tests verifying the correctness of individual key derivation steps are available on Github. See TEST-7 in the index of unit tests for a static repository link.

3.4.2 INV-21 — Key derivation algorithm diverges from specification

The key derivation protocol implemented by the target release differs from the protocol specification. The public key "diversifier" described in the specification is removed in the implementation in favor of a single public key `pk_d` associated with each secret key `sk`.

This finding does not pose a security risk.

Recommendation. Update the specification document to reflect the modified key derivation design.

3.4.3 INV-22 — Keyspace overlap with Zcash Sapling shielded pool

The key derivation protocol implemented by the target release uses Sapling default personalization for several steps which could result in overlap between Zcash and Iron Fish address spaces. This could be a compatibility issue for cross-chain bridging of assets between these networks.

This finding does not currently pose a security risk.

Description. An "expanded" key with spend authorization key `ask` and proof authorization key `nsk` leads to the same incoming viewing key `ivk` in both the Iron Fish and Sapling key derivation protocols.

Iron Fish key derivation uses a hard-coded generator `PUBLIC_KEY_GENERATOR` to produce a public address `pk_d` from the incoming viewing key `ivk`. This generator was chosen randomly according to communication with the development team, but this fact does not appear to be well-documented. This random generator selection implies that Iron Fish public addresses have only trivial risk of collision with Sapling public addresses.

Recommendation. Ensure that personalized generators are used in place of the default values of G_{ak}, G_{nk}, and g_d. Document how these personalized generators were chosen, so that network participants can be confident of the independence of group generators used at different stages of key derivation. Additionally, use non-default personalization for the Blake2s hash evaluation used to generate the incoming view key `ivk` from `ak` and `nk`.

Note. The recommended changes are not essential for secure operation of Iron Fish as an independent network, but may be important for interoperability with the Zcash Sapling shielded pool at some point in the future, and so may be considered as an upgrade when such functionality is required.

3.4.4 INV-23 — Correctness of key derivation

The key derivation algorithm implemented in the target release was checked for correctness.

No issue was found.

3.5 Client Validation and Consensus

3.5.1 INV-24 — Pure mint transactions are arbitrarily replayable

A transaction which mints a custom asset, spends no notes, and specifies zero network fee is allowed by node functioning and network consensus rules, and may be replayed on the blockchain multiple times.

This issue has been fixed.

Description. We define a “pure mint” transaction to be a transaction which mints some amount of a custom Iron Fish asset, outputs the minted coins to one or more addresses, spends no existing notes, and specifies a network fee of zero. Such a transaction could plausibly be considered valid for a number of purposes (for instance, if a miner has an agreement with another network operator to include zero-fee mints of a bridged asset), and in particular, the implementation of node logic

and network consensus rules on `v0.1.76` clients supports the inclusion of such transactions on the blockchain.

This functionality leads to a vulnerability due to a mismatch with the security assumptions made by the underlying Sapling transaction design. Mainly, the mechanism used to prevent transactions from being replayed on the blockchain is the list of *nullifiers* generated when spending notes. A transaction which spends a note produces a global nullifier uniquely associated with that note, so any attempt to reuse the transaction verbatim will fail because the transaction produces the same nullifier upon re-execution, resulting in rejection from chains that have already recorded this nullifier.

A pure mint transaction spends no notes, and so produces no nullifiers. As a result, the network consensus rules do not detect when such a transaction has already been included on the blockchain, and so it can be added to the chain multiple times. In particular, since no additional authentication is required to submit a previously posted transaction to the network, this means that an unauthorized user can cause a pure mint transaction to be re-executed on chain without permission from the transaction's originator.

Recommendation. Several approaches could resolve this vulnerability. If pure mint transactions are determined to be unwanted behavior on the network, then it is enough to modify the network consensus logic to reject such transactions, either by requiring that all transactions spend at least one note, or by specifying that transaction fees must be positive instead of only non-negative.

To mitigate the replay vulnerability while preserving the validity of pure mint transactions, a further record of transactions included on the chain would be necessary. One way to do this would be to add a random plaintext "serial number" to mint operations which is treated similarly to standard note nullifiers and stored by nodes in a separate database. Another would be to maintain a database of transaction hashes for all transactions included on the blockchain, against which new transactions could be compared.

A separate recommendation, though not a substitute to the validations mentioned above, would be to implement a mechanism for issuer key rotation of assets issued on chain. This is a good way to manage the risk of assets.

Tests. Unit tests were written verifying the above noted behaviors on a `v0.1.76` base. See TEST-4, TEST-5, and TEST-6 in the index of unit tests for more details and static repository links.

Fixed. This issue was first reported to the Iron Fish team on Thursday, May 4th. As of version `v1.3.0`, this issue has been fixed. The Iron Fish team implemented a consensus rule that only strictly higher-than-zero fees are allowed in transactions. See Pull Request 3901 for more details.

3.5.2 INV-25 — Inline double-spend transactions allow memory pool contamination

Transactions which attempt to spend the same note multiple times in a single transaction are accepted into a node's mempool when received from a peer node, and are incorporated into mining block templates from a node's mempool.

This issue has been fixed.

Description. We define an "inline double-spend" transaction to be any transaction which includes more than one spend operation with the same nullifier. Such a transaction is never valid, and should be rejected from network operations when detected. In release `v0.1.76`, an inline double-spend transaction received from a network peer will be accepted by a node and added to its mempool. The following code in the mempool procedure to add a transaction is the source of this issue, and results in the observed behavior because the nullifiers for a transaction are added to the list of currently known nullifiers as a batch *after* checking all of the nullifiers in the transaction for double-spends.

```
// excerpt from: ironfish/src/mempool/memPool.ts:271-290
for (const spend of transaction.spends) {
  const existingHash = this.nullifiers.get(spend.nullifier)
  const existingTransaction = existingHash && this.transactions.get(existingHash)

  if (!existingTransaction) {
    continue
  }
  // ...
}
// ...
for (const spend of transaction.spends) {
  this.nullifiers.set(spend.nullifier, hash)
}
```

Once a transaction has been added to a node's memory pool, there is limited further verification of transaction validity. In particular, an inline double-spend transaction in the mempool will be included in newly constructed block templates meant to be sent to miners, producing templates that are not suitable for mining. Note however that the resulting invalid block templates *are* detected prior to being submitted to the miner RPC, due to a final check of candidate blocks against the real network consensus rules.

The above behavior produces two significant vulnerabilities for proper node functioning. First, a miner node which receives an inline double-spend transaction from the network with a high fee will include that transaction in subsequent requests for new miner block templates, and so will cease to produce valid templates for miners. This subsequently causes miners to cease operation due to lack of templates to mine on.

Second, since inline double-spend transactions will never be included in valid blocks, there is no

mechanism to purge such transactions from node mempools once they are received from the network. This opens an avenue for a denial of service attack, where a node is presented many invalid transactions of this form which are added to the mempool, wasting memory resources on the host machine.

Recommendation. Modify the mempool verification logic to detect and correctly reject inline double-spend transactions. More generally, use full network consensus logic rather than ad-hoc checks when screening transactions which are to be added to the mempool to avoid situations where the mempool can contain transactions which are not valid on the blockchain.

Tests. Unit tests were written verifying the above noted behaviors on a `v0.1.76` base. See TEST-1, TEST-2, and TEST-3 in the index of unit tests for more details and static repository links.

Fixed. This issue was first reported to the Iron Fish team on Thursday, May 4th. As of version `v1.2.0`, this issue has been fixed. The Iron Fish implemented the recommended fix that the client verifies whether there are the same nullifiers in the same transaction, before it is allowed in the mempool. See Pull Request 3898 for more details.

3.5.3 INV-26 — Insufficient validation of mempool transactions

There is a broader risk of Denial of Service (DoS) when it comes to more generically flooding the mempool with invalid transactions.

This finding does not currently pose a security risk.

Description. The mempool transaction validation process is not stateful and does not check against the current state of the chain. This means that certain transaction, though invalid, could be accepted into the mempool. The actual consensus validation checks against the state and those transactions would not be added to the blockchain state.

A high-fee transaction could motivate broadasts to happen for invalid transactions, flooding the mempool.

Recommendation. It would be ideal to get the mempool validation to check against the state and it should use functions such as `verifier.verifyBlockConnect`.

3.5.4 INV-27 — No validation of transactions received from the API

Nodes are more permissive for transactions received from the API than for those received from the peer-to-peer network.

This finding does not pose a security risk.

Description. The node validation design makes use of the security model that API operations executed locally are fully trusted, and therefore does little validation on such operations. However, it could be useful in some cases to incorporate full validity checks (those done on transactions received from the peer-to-peer network) on API calls, at least as an option which can be enabled by an execution flag. For instance, a node operator might wish to deploy their node in such a way that API access is made available to the internet or behind a backend service, in which case it would be convenient to allow such checks to be enabled on demand.

3.5.5 INV-28 — Confusing naming in Spend.commitment attribute

The attribute `Spend.commitment` from file `ironfish/src/primitives/spend.ts` actually refers to a Merkle tree root for the note commitment tree. The current attribute name could lead to confusion.

This finding does not currently pose a security risk.

3.5.6 INV-29 — Clients reject network transactions with negative transaction fee

It was verified that client software correctly rejects transactions received from network peers which specify a negative transaction fee.

No issue was found.

Tests. A unit test was written which confirms correct functioning of client software in this case. See TEST-9 in the index of unit tests for more details and a static repository link.

3.6 General Findings

3.6.1 INV-30 — Unreliable fixtures in unit tests

Some unit tests in older client versions fail when related testing fixtures are not previously generated, but then pass once the fixtures have been generated. This could indicate an issue with the logic of the testing framework setup.

This finding does not currently pose a security risk.

Description. On the `v0.1.62` Iron Fish client release, it was discovered that some tests which pass when executed normally will fail when related TypeScript fixtures are deleted and the test is forced to run without cached values. After failing once and regenerating fixtures, the tests return to passing when run again. Three particular tests which exhibited this behavior can be found in the file `ironfish/src/network/peerNetwork.test.ts`, with descriptions:

- "does not accept or sync transactions when the worker pool is saturated"
- "does not accept or sync transactions when the node is syncing"
- "does not sync or gossip double-spent transactions"

The behavior can be reproduced by increasing the timeout parameter for the Jest testing framework from `-5000` to `-10000` in file `ironfish/package.json:54`, deleting the fixtures directory `ironfish/src/network/__fixtures__/`, and then running the peer network tests from the `ironfish` subdirectory via `yarn run test src/network/peerNetwork.test.ts`.

This behavior was no longer observed for this file on the `v0.1.76` client release. Due to time constraints of the review, it was not checked whether this issue still is present elsewhere in this release.

Recommendation. Verify that the use of fixtures in unit tests does not conceal test failures.

3.6.2 INV-31 — Rust code implemented in Node-API wrapper is untested

Non-trivial cryptographic functionality is implemented in the `ironfish-rust-nodejs` repository, which is untested due to technical limitations surrounding the use of the `napi-rs` crate to construct the Node wrapper around native Rust code.

This finding does not currently pose a security risk.

Recommendation. Refactor so that all non-trivial Rust code currently implemented in the `ironfish-rust-nodejs` repository is moved to the `ironfish-rust` repository. Ideally, each wrapped function call in the Node-API wrapper layer should be a simple call to a corresponding Rust function imported from `ironfish-rust`. Implement unit tests for all refactored functionality.

3.6.3 INV-32 — Iron Fish whitepaper diverges from implementation

The whitepaper describing the technical workings of the Iron Fish protocol is out of sync with the features implemented in the Iron Fish client software.

This finding does not pose a security risk.

Description. The Iron Fish whitepaper (Iron Fish Team 2020) describes in significant detail the cryptographic and peer-to-peer protocols which are utilized to provide a secure, private blockchain client. While this document is a useful reference for what it describes, the content has fallen out of sync with the project's current implementation and feature set. Parts of the whitepaper which do not match the implementation by the software client include:

- The whitepaper does not discuss the implementation of custom assets, and the corresponding Mint and Burn transaction types.
- The whitepaper does not discuss the addition of the public address of a note's sender to the plaintext data of notes, nor the changes to SNARK proofs and transaction verification supporting this feature.
- The whitepaper still refers to key "diversifiers", a feature of the Zcash Sapling key structure which has been eliminated from the current Iron Fish design.

This is an issue both from the standpoint of providing technical clarity about features provided by the client, and of providing community members with an accurate and up-to-date representation of the protocol's basic functionality.

Recommendation. Revise the Iron Fish whitepaper so that it matches the current design implemented by the network client. Consider being more explicit about which client release the whitepaper specification is consistent with; for instance, include an indication of a release version and date. Also consider maintaining previous and current revisions of the whitepaper in a resilient and self-contained file format (e.g. PDF) so that community members can reliably refer to historical versions of the document.

3.6.4 INV-33 — Iron Fish protocol lacks a formal security specification

There does not appear to be any published resource formally specifying the concrete security properties the Iron Fish protocol is meant to satisfy. Without such a security specification, it is difficult to reason formally about the overall security of the protocol.

This finding does not currently pose a security risk.

Description. The Iron Fish protocol whitepaper (Iron Fish Team 2020) provides a detailed design for a "decentralized" and "censorship-resistant" blockchain supporting "strong privacy guarantees". However, these terms, while suggestive, do not provide a detailed enough description of the security and privacy properties required of the protocol to assess formally whether the design can be said to satisfy such properties.

One example of an instance where a more detailed specification would help to inform a rigorous security analysis is the following. In the current Iron Fish design, a note's sender is required to include their public address in the plaintext data of each note they send. This is a nice feature both from a usability and a compliance standpoint, but it also introduces a form of "weak linkability" to Iron Fish transactions: any network participant can verifiably prove the source of the assets coming from any transaction they receive. This is in contrast to other privacy oriented protocol designs which do not allow any network participant to unilaterally reveal information about the network activity of another participant, and opens the protocol up to some restricted forms of network analysis that could violate naive expectations of "strong privacy guarantees".

This is a valid design choice for the network's operation, and is a potentially useful trade-off sacrificing a small amount of privacy in favor of an in-band usability and compliance feature. However, it is not clear whether the design preserves the promised strong privacy guarantees, because these guarantees are never explicitly stated. Providing a careful specification of the required security and privacy properties gives clarity to both users and developers about the behavior of the network.

Recommendation. Publish a formal specification of the security and privacy properties that the Iron Fish protocol is meant to satisfy, and conduct an analysis confirming that the implemented protocol satisfies these properties.

3.6.5 INV-34 — Missing framework requirements from Zcash imports

There are some imports that would be useful to add to the code.

This finding does not currently pose a security risk.

Recommendation. It is recommended to add the following set of imports.

In `bls12_381`:

- `add_assign` and `double` functions

In `bellman`:

- `AllocatedNum::inputize, to_bits_le_strict`
- `Boolean, AllocatedBit::alloc`

In `zcash_proofs`:

- `Version 0.7.1`
- `Cargo.lock`: `98bf5f6af051dd929263f279b21b9c04c1f30116c70f3c190de2566677f245ef`
- `EdwardsPoint::repr, fixed_base_multiplication, mul, inputize`
- `ecc::fixed_base_multiplication`
- `to_bits_le_strict` (forbid congruency)
- `generate_circuit_generator` (table for fixed-base `EC_MUL`)
- `constants::*`
- `fixed_base_multiplication, FixedGenerator`, which must have the right size (unchecked because of `.zip()`)

In `zcash_primitives`:

- `Version 0.7.1`
- `Cargo.lock`: `4fbb401f5dbc482b831954aaa7cba0a8fe148241db6d19fe7cebda78252ca680`
- `constants::*`

3.6.6 INV-35 — Client `addTransaction` API has unintuitive behavior

In the current implementation, when `addTransaction` returns `true`, it does not immediately imply that the transaction is accepted into the mempool, as it could be evicted immediately. Even if a transaction does not stay in this node's mempool, the transaction is still broadcasted to peers, and the API client still receives "`accepted = true`" as a response.

This finding does not pose a security risk.

3.6.7 INV-36 — Handling of non-unique representations for curve points and scalars

Algebraic primitives were checked for common problems involving non-unique representations. There are no issues with non-unique representation of points, as the co-factors have been cleared for elliptic curve points, and there are no issues with non-unique representations of elements, as no overflows are allowed for field elements and points in the curve.

No issue was found.

3.6.8 INV-37 — Correct witness generation process

It was verified that witnesses are generated correctly for each of the implemented proof circuits, so that each zk-SNARK proof is constructed using the trace of the program constraints to be verified on a concrete instance.

No issue was found.

4 Bibliography

Hopwood, Daira, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2016. "Zcash Protocol Specification." *GitHub: San Francisco, CA, USA*. https://zips.z.cash/protocol/protocol.pdf.

Iron Fish Team. 2020. "Iron Fish Whitepaper." *Online*. https://ironfish.network/learn/whitepaper/introduction.