Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

.arianee

Full-Privacy Extension Smart Contracts

Veridise Inc.

July 17, 2024

► **Prepared For:**

Arianee
https://www.arianee.org/

► **Prepared By:**

Jon Stephens
Ian Neal
Mark Anthony

► **Contact Us:**

contact@veridise.com

► **Version History:**

July 17, 2024	V2
July 16, 2024	V1
July 10, 2024	Initial Draft

Contents

1 Executive Summary

From July 8, 2024 to July 10, 2024, Arianee engaged Veridise to review the security of their Full-Privacy Extension Smart Contracts. The review covered both the smart contracts and zero-knowledge circuits that allow issuers to privately interact with the Arianee protocol. Note, however, that this report will focus on the smart contract review. Veridise conducted the assessment over 9 person-days, with 3 engineers reviewing code over 3 days on commit `7b6e47a`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual code review.

Project summary. Arianee allows brands to issue and manage NFTs for a variety of purposes. This review did not cover the entire protocol, but rather is limited to their full-privacy extension. Specifically, this report will focus on the associated smart contracts.

The in-scope contracts allow issuers to manage their NFTs while keeping the identity of the issuer private. To do so, issuers must first privately purchase credits with the help of the CreditRegistrationProof ZK Circuit. Purchasing credits will associate a credit balance with a public commitment that is placed in a Merkle Tree. This allows issuers to later spend those credits as they manage the NFTs using a CreditNoteProof. This proof is generated with a ZK Circuit that proves an issuer knows the secret information used to create a commitment that is contained in the on-chain Merkle Tree. A single CreditNoteProof can only be spent 1000 times as it can only be associated with 1000 unique nullifies.

Once an issuer has credits they can spend them to privately issue or manage NFTs. They do so by interacting with the ArianeeIssuerProxy which associates NFT issuers with private commitments. When an issuer creates an NFT, they will generate an OwnershipProof using a ZK circuit which will prove that they know the secret information required to craft the commitment. After creation, Ownership proofs will be used to prove that the issuer knows how to create the commitment associated with the NFT so that they can perform management tasks provided by the ArianeeStore and ArianeeSmartAsset.

Code assessment. The Arianee developers provided the source code of the Arianee contracts for review. The source code appears to be mostly original code written by the Arianee developers. It contains some documentation in the form of READMEs and documentation comments on some functions and most storage variables. To facilitate the Veridise auditors' understanding of the code, the Arianee developers also shared documentation that describes the intended user-flows in the Full-Privacy Extension Smart Contracts and the Veridise auditors referred to publicly-available Arianee documentation as necessary. The source code contained a test suite, which tested the main user-flows of the project and included several negative-tests as well.

Summary of issues detected. The audit uncovered 17 issues, 2 of which are assessed to be of high or critical severity by the Veridise auditors. Specifically, the `reserveToken` mechanism can be used to DoS the proxy contract and the Arianee SmartAsset store (V-ARIA-VUL-001), and

that previous smart asset owners can steal assets after selling them since the `tokenAccess` key is not reset on transfer (V-ARIA-VUL-002). The Veridise auditors also identified 1 medium-severity issue, in that invalid commitment hashes may permanently lock token IDs in the proxy contract (V-ARIA-VUL-003), as well as 5 low-severity issues, 8 warnings, and 1 informational finding. The Full-Privacy Extension Smart Contracts developers have been notified of the issues and have indicated an intent to fix them.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve the Full-Privacy Extension Smart Contracts.

Document Best-Practices. While the privacy extension could allow issuers to privately interact with the protocol, it is possible that some actions could expose information about them if they are not careful (see V-ARIA-VUL-006 and V-ARIA-VUL-008). We would recommend that Arianee developers create a best-practices page which explains how issuers should protect their information (not re-using token access keys, using popular interface providers, not re-using commitments, ensuring no identifiable information is publicly observable in created NFTs, using relayers, etc.).

Clarify Licensing. Some of the contracts contain an SPDX License Identifier of "UNLICENSED" (meaning third parties are not allowed to use the code, all rights reserved), while others are "MIT" licensed (which is a permissive open-source license). Furthermore, the repository as a whole, which is publicly viewable, has no LICENSE file, which generally indicates that all contained source code is not open-source (same semantics as "unlicensed"). We recommend clarifying the open-source status of the project with an explicit statement (be that an open-source license or an "all rights reserved" statement) to clarify what viewers of the repository are allowed to do with Arianee's code.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Full-Privacy Extension Smart Contracts	`7b6e47a`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
July 8–July 10, 2024	Manual & Tools	3	9 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	1	1	0
High-Severity Issues	1	1	0
Medium-Severity Issues	1	1	0
Low-Severity Issues	5	4	2
Warning-Severity Issues	8	7	4
Informational-Severity Issues	1	1	1
TOTAL	17	15	7

Table 2.4: Category Breakdown.

Name	Number
Data Validation	4
Access Control	2
Information Leakage	2
Denial of Service	1
Logic Error	1
Gas Optimization	1
Missing/Incorrect Events	1
Centralization	1
Usability Issue	1
Replay Attack	1
Dead Code	1
Maintainability	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Full-Privacy Extension Smart Contracts's smart contracts. In our audit, we sought to answer questions such as:

- Is information leaked about the issuer as they interact with the Arianee protocol?
- Can a brand be identified as a private issuer?
- Can credit notes be spent without the permission of the purchasing account?
- Can a private issuer impact the user experience of other users?
- Can the privacy extension violate assumptions made by the ArianeeStore?
- Is it possible to replay a ZK proof?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool, Vanguard. This tool is designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.

Scope. The scope of this audit is limited to the following files in the `https://github.com/Arianee/ArianeeMaster` repository:

- `contracts/ArianeePrivacy/ArianeeIssuerProxy.sol`
- `contracts/ArianeePrivacy/ArianeeCreditNotePool.sol`

Methodology. Veridise auditors inspected the provided tests, and read the Full-Privacy Extension Smart Contracts documentation. They then began a manual review of the code assisted by static analysis. At the start of the audit, Veridise auditors met with Arianee developers to discuss the project. During the course of the audit, they also asked them questions in a shared Slack channel as necessary.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-ARIA-VUL-001	reserveToken can be used to DoS the . . .	Critical	Acknowledged
V-ARIA-VUL-002	Previous owners can steal smart assets	High	Acknowledged
V-ARIA-VUL-003	updateCommitment may accept invalid . . .	Medium	Acknowledged
V-ARIA-VUL-004	Gas Optimizations	Low	Fixed
V-ARIA-VUL-005	Users can avoid ArianeeStore whitelist	Low	Intended Behavior
V-ARIA-VUL-006	TokenAccess can leak issuer identity . . .	Low	Acknowledged
V-ARIA-VUL-007	Event not emitted on important update	Low	Fixed
V-ARIA-VUL-008	_interfaceProvider could leak information	Low	Acknowledged
V-ARIA-VUL-009	wordPos max range is unchecked	Warning	Fixed
V-ARIA-VUL-010	creditNotePools cannot be removed	Warning	Intended Behavior
V-ARIA-VUL-011	Unchecked approve return value	Warning	Fixed
V-ARIA-VUL-012	Centralization Risk	Warning	Acknowledged
V-ARIA-VUL-013	Missing address zero-checks	Warning	Acknowledged
V-ARIA-VUL-014	Token ID reuse can have undesired side- . . .	Warning	Acknowledged
V-ARIA-VUL-015	OwnershipProofs may be replayed . . .	Warning	Fixed
V-ARIA-VUL-016	Unused Code and Dependencies	Warning	Fixed
V-ARIA-VUL-017	Typos and incorrect comments	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-ARIA-VUL-001: reserveToken can be used to DoS the protocol

Severity	Critical	**Commit**	7b6e47a
Type	Denial of Service	**Status**	Acknowledged
File(s)	`ArianeeIssuerProxy.sol`		
Location(s)	reserveToken		
Confirmed Fix At	N/A		

The `reserveToken` function, which is callable by anyone, does not check if the provided `_commitmentHash` is a valid commitment hash, as the `tryRegisterCommitment` function only checks if a commitment hash has been added for a given token, not if that commitment hash is a valid commitment.

```
function tryRegisterCommitment(uint256 _tokenId, uint256 _commitmentHash) internal {
  require(
    commitmentHashes[_tokenId] == 0,
    'ArianeeIssuerProxy: A commitment has already been registered for this token'
  );
  commitmentHashes[_tokenId] = _commitmentHash;
}

// VERIDISE: ...

function reserveToken(uint256 _commitmentHash, uint256 _tokenId) external {
  tryRegisterCommitment(_tokenId, _commitmentHash);
  store.reserveToken(_tokenId, address(this));
}
```

Snippet 4.1: Snippet from `ArianeeIssuerProxy`.

Impact **Denial-of-Service Attack.** Any user can front-run another user's `reserveToken` transaction with a random value provided for `_commitmentHash`, locking the desired `_tokenId`. Since changing a token's commitment hash requires an ownership proof (via `updateCommitment`), front-running `reserveToken` with an arbitrary value for `_commitmentHash` will lock that `_tokenId`, as updating the commitment would require being able to reverse the commitment hash function. Furthermore, since `reserveToken` reserves the token in the `store`, this DoS attack can additionally deny service to the store, not just the issuer proxy.

Possible User Errors. If a non-malicious user accidentally provides the wrong value for `_commitmentHash`, this function will succeed, but will permanently lock the desired token ID as described above. This causes users to waste time and gas re-submitting a new `reserveToken` request with a new, unreserved `_tokenId` argument.

Recommendation To fix the permanent lock-up of `_tokenIds` behind invalid `_commitmentHashs`, we recommend performing an ownership proof check after the token ID is reserved.

This would validate that the sender has sent a valid `_commitmentHash`.

```
function reserveToken(OwnershipProof calldata _ownershipProof, uint256
    _commitmentHash, uint256 _tokenId) external {
  tryRegisterCommitment(_tokenId, _commitmentHash);
  store.reserveToken(_tokenId, address(this));
  // VERIDISE: added proof verification
  _verifyProof(_ownershipProof, false, _tokenId);
}
```

Snippet 4.2: Suggested fix for validating the `_commitmentHash` in `reserveToken`.

However, even with the above fix, it is still possible to launch a DoS attack on the protocol, as an attacker could also craft a valid call to `reserveToken` with the same `_tokenId` as another user. To prevent such DoS attacks, we recommend limiting the audience of who can use this function.

Developer Response We are aware of this problem, and it exists even outside the 'Full-Privacy' context. We are waiting for some business decisions to address this issue, and we will patch the `ArianeeStore` contract *(and make some change to the `ArianeeIssuerProxy` accordingly if needed)* once those decisions are made.

We will implement one of the following three solutions in the future:

- Require credit payment at the time of token reservation to make the attack economically unviable.
- Assign a token ID dynamically based on the `msg.sender` (coupled with any source of provable randomness) to prevent an external attacker from front-running the reservation.
- Allow token reservation only for certain Issuers.

4.1.2 V-ARIA-VUL-002: Previous owners can steal smart assets

Severity	High	**Commit**	7b6e47a
Type	Logic Error	**Status**	Acknowledged
File(s)	`ArianeeSmartAsset.sol`		
Location(s)	_transferFrom		
Confirmed Fix At	N/A		

It should be noted that this is outside the scope of the current audit and was discovered while investigating a potential issue that was in-scope. The `ArianeeSmartAsset` allows an operator of the given token to define a token access key (called `tokenAccess`) that gives anyone with knowledge of the key special abilities. One such ability that may be granted allows the token to be transferred to a new owner by providing a message signed with the given key. Notably, however, when a token is transferred, the token access key is not reset as shown below.

```
function _transferFrom(address _from, address _to, uint256 _tokenId) internal
    override {
  require(store.canTransfer(_from, _to, _tokenId, isSoulbound), "ArianeeSmartAsset:
    Transfer rejected by the store");

  if (isSoulbound) {
    address tokenOwner = idToOwner[_tokenId];
    require(tokenOwner == _from, NOT_VALID_NFT);

    address tokenIssuer = certificate[_tokenId].tokenIssuer;

    // If the owner is NOT the issuer, the token is soulbound and the transfer can be
     made only by the issuer to change the owner if needed
    if (tokenOwner != tokenIssuer) {
      require(tokenIssuer == _msgSender(), "ArianeeSmartAsset: Only the issuer can
    transfer a soulbound smart asset");
    }

    /*
    * If the previous condition has not been hit, the owner IS the issuer and the
    token is not soulbound yet or not anymore for a limited time.
    * This is either the first transfer of the token to its first "real" owner or a
    recovery request made by the issuer on the behalf of the owner (i.e the owner
    lost his wallet and wants to recover his token)
    */
  }

  super._transferFrom(_from, _to, _tokenId);
  arianeeWhitelist.addWhitelistedAddress(_tokenId, _to);

  if (_isFirstTransfer(_tokenId)) {
    idToFirstTransfer[_tokenId] = false;
    store.dispatchRewardsAtFirstTransfer(_tokenId, _to);
  }
}
```

Snippet 4.3: Overridden definition of the `_transferFrom` function which does not reset `tokenAccess`.

Impact Since the `tokenAccess` is not reset, a SmartAsset operator could sell the token, transfer it to the new owner and later recover the asset while keeping the paid funds.

Recommendation Delete or reset the `tokenAsset` upon a transfer.

Developer Response We have noted this issue and will address it in a future sprint, as it is currently out of scope for now. However, we appreciate your diligence in bringing this to our attention.

4.1.3 V-ARIA-VUL-003: updateCommitment may accept invalid commitment hash

Severity	Medium	**Commit**	7b6e47a
Type	Data Validation	**Status**	Acknowledged
File(s)	`ArianeeIssuerProxy.sol`		
Location(s)	updateCommitment		
Confirmed Fix At	N/A		

The `updateCommitment` function updates a token ID's commitment by first verifying that the provided ownership proof is a valid proof for the current commitment, then updates the commitment to a new commitment hash. However, there is no check that the `_newCommitmentHash` provided to `updateCommitment` is a valid commitment hash.

```
function updateCommitment(OwnershipProof calldata _ownershipProof, uint256 _tokenId,
    uint256 _newCommitmentHash) public onlyWithProof(_ownershipProof, false, _tokenId
    ) onlyOwner {
  require(
    commitmentHashes[_tokenId] != 0,
    'ArianeeIssuerProxy: No commitment registered for this token'
  );
  commitmentHashes[_tokenId] = _newCommitmentHash; // VERIDISE: this could be an
    arbitrary value that is not a valid commitment hash
}
```

Snippet 4.4: Snippet showing `updateCommitment()` from `ArianeeIssuerProxy`.

Impact If the user makes a mistake and provides an invalid hash for `_newCommitmentHash`, the token will be locked behind that commitment hash, as they will be unable to create an ownership proof that would allow them to update the commitment again.

Recommendation Validate that the `_newCommitmentHash` is valid by checking a pre-update ownership proof and a post-update ownership proof, which will validate that the proof creator is the owner of both the old credentials and new credentials.

Developer Response We believe that requiring an additional proof when calling `updateCommitment` is too cumbersome. We will ensure on the SDK side (during the implementation of this feature) to minimize the possibility of passing an invalid commitment. All our users use our SDK to interact with the protocol, so it's very unlikely that someone would interact directly with it.

4.1.4 V-ARIA-VUL-004: Gas Optimizations

Severity	Low	**Commit**	7b6e47a
Type	Gas Optimization	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	7086e42		

In the following locations, the auditors identified missed opportunities for potential gas savings:

- ArianeeCreditNotePool.sol:
  - The addresses for `issuerProxy`, `token`, `store`, `creditRegister`, and `creditVerifier` are only set in the constructor; these fields could be made `immutable`, which would save gas.
- ArianeeIssuerProxy.sol
  - The addresses for `store`, `smartAsset`, `arianeeEvent`, `arianeeLost`, and `verifier` are only set in the constructor; these fields could be made `immutable`, which would save gas.

Impact Gas may be wasted, costing users extra funds.

Recommendation Perform the optimizations.

Developer Response All unmodifiable addresses have been made `immutable`, as per the recommendation.

4.1.5 V-ARIA-VUL-005: Users can avoid ArianeeStore whitelist

Severity	Low	Commit	7b6e47a
Type	Access Control	Status	Intended Behavior
File(s)	`ArianeeCreditNotePool.sol`, `ArianeeIssuerProxy.sol`		
Location(s)	purchase, reserveToken		
Confirmed Fix At	N/A		

The `ArianeeIssuerProxy` and `ArianeeCreditNotePool` are intended to allow issuers to privately interact with the `ArianeeStore` contract. The `ArianeeStore` only allows certain actions to be performed by whitelisted addresses, such as the `buyCredit` action shown below.

```
function buyCredit(uint256 _creditType, uint256 _quantity, address _to) external
    onlyAllowedBuyer(_to) whenNotPaused() {
    uint256 tokens = _quantity * creditPrices[_creditType];

    // Transfer required token quantity to buy quantity credit
    require(acceptedToken.transferFrom(
            _msgSender(),
            address(this),
            tokens
        ));

    creditHistory.addCreditHistory(_to, creditPrices[_creditType], _quantity,
    _creditType);

    emit CreditBought(_msgSender(), _to, _creditType, _quantity);
}
```

Snippet 4.5: Definition of the `buyCredit` function.

Both the `ArianeeIssuerProxy` and `ArianeeCreditNotePool` invoke whitelisted functions and so they must be whitelisted to function properly. They, however, do not whitelist their users in any way as shown below.

```
function purchase(CreditRegistrationProof calldata _creditRegistrationProof, bytes32
    _commitmentHash, uint256 _zkCreditType) external nonReentrant {
    require(
        _zkCreditType >= 1 && _zkCreditType <= 4,
        'ArianeeCreditNotePool: The credit type should be either 1, 2, 3 or 4'
    );
    require(
        !commitmentHashes[_commitmentHash],
        'ArianeeCreditNotePool: This commitment has already been registered'
    );

    _verifyRegistrationProof(_creditRegistrationProof, _commitmentHash, _zkCreditType
    );

    uint32 insertedIndex = _insert(_commitmentHash);
    commitmentHashes[_commitmentHash] = true;

    // The credit type is 0-indexed in the store, but 1-indexed in the commitment
    uint256 creditType = _zkCreditType - 1;
    uint256 creditPrice = store.getCreditPrice(creditType);

    // One credit note is worth `MAX_NULLIFIER_PER_COMMITMENT` credits
    uint256 amount = MAX_NULLIFIER_PER_COMMITMENT * creditPrice;

    // The caller should have approved the contract to transfer the amount of tokens
    token.safeTransferFrom(_msgSender(), address(this), amount);

    // Approve the store to transfer the required amount of tokens
    token.approve(address(store), amount);
    // Buy the credits from the store
    store.buyCredit(creditType, MAX_NULLIFIER_PER_COMMITMENT, issuerProxy);

    emit Purchased(creditType, _commitmentHash, insertedIndex, block.timestamp);
}
```

Snippet 4.6: Definition of the `purchase` function. Note that `_verifyRegistrationProof` effectively just verifies the `CreditRegistrationProof`.

Impact Users may use the `ArianeeIssuerProxy` to overcome a whitelist in the `ArianeeStore` and so anyone can interact with the `ArianeeStore`.

Recommendation If the intention is to only allow approved issuers to interact with the `ArianeeStore`, additional validation must be performed. As an example, Arianee could maintain a Merkle tree with approved secret commitments. To access any whitelisted functionality a user must then prove their existence in the tree.

Developer Response The `onlyAllowedBuyer` modifier is used only in specific deployments called "dedicated". All "Full-Privacy" deployments will not be dedicated, so we can safely ignore this. Everyone should be able to call the ArianeeStore.

4.1.6 V-ARIA-VUL-006: TokenAccess can leak issuer identity information

Severity	Low	**Commit**	7b6e47a
Type	Information Leakage	**Status**	Acknowledged
File(s)	`ArianeeIssuerProxy.sol`		
Location(s)	addTokenAccess, hydrateToken		
Confirmed Fix At	N/A		

A SmartAsset may be associated with a token access key that gives anyone with knowledge of this key special abilities. The `ArianeeIssuerProxy` allows issuers to create such a key using the `addTokenAccess` function shown below. As an issuer is intended to be private, if the same access key is used multiple times or is associated with the brand at any point in the future, information could be learned about the issuer.

```
function addTokenAccess(
  OwnershipProof calldata _ownershipProof,
  uint256 _tokenId,
  address _key,
  bool _enable,
  uint256 _tokenType
) external onlyWithProof(_ownershipProof, false, _tokenId) {
  smartAsset.addTokenAccess(_tokenId, _key, _enable, _tokenType);
}
```

Snippet 4.7: Definition of the `addTokenAccess` function.

Impact The token access key could be used to learn information about an issuer or a brand.

Recommendation We would recommend creating a best-practices page for brands interacting with the `ArianeeIssuerProxy` contract and recommend that they use unique keys if they use this API.

Developer Response Brands already use unique keys for each of their tokens. We will ensure this continues with this new version.

4.1.7 V-ARIA-VUL-007: Event not emitted on important update

Severity	Low	**Commit**	7b6e47a
Type	Missing/Incorrect Events	**Status**	Fixed
File(s)	`ArianeeIssuerProxy.sol`		
Location(s)	Multiple		
Confirmed Fix At	9c5f6f0		

It is often considered a best practice to emit an event upon a state update to indicate that a change was made. Doing so allows both external users and protocol administrators to monitor the protocol for a variety of reasons, including for potentially suspicious activity. It is therefore critical for significant changes to the protocol to be accompanied with events to enable this monitoring. The following functions perform important updates in the proxy contract but do not emit an event from the proxy:

- `addCreditNotePool`
- `addCreditFreeSender`
- `removeCreditFreeSender`
- `reserveToken`
- `hydrateToken`
- `updateCommitment`

Impact These variables modify important state and changes to them can indicate issues within the protocol.

Recommendation Emit events upon important state updates.

Developer Response The developers added the following events: `CreditFreeSenderAdded`, `CreditFreeSenderRemoved`, `CreditNotePoolAdded`, `TokenCommitmentRegistered`, `TokenCommitmentUpdated`, and `TokenCommitmentUnregistered`.

4.1.8 V-ARIA-VUL-008: _interfaceProvider could leak information

Severity	Low	**Commit**	7b6e47a
Type	Information Leakage	**Status**	Acknowledged
File(s)	`ArianeeIssuerProxy.sol`		
Location(s)	hydrateToken, updateSmartAsset, createEvent, createMessage		
Confirmed Fix At	N/A		

Several functions, such as the one below, allow an issuer to specify an interface provider who receives a reward from the `ArianeeStore`. According to the developers, this is to reward projects that help brands build on Arianee. If a single or a small number of private brands were associated with an interface provider, however, it is possible that a brand's identity could be revealed.

```
function updateSmartAsset(
  OwnershipProof calldata _ownershipProof,
  CreditNoteProof calldata _creditNoteProof,
  address _creditNotePool,
  uint256 _tokenId,
  bytes32 _imprint,
  address _interfaceProvider
) external onlyWithProof(_ownershipProof, true, _tokenId) {
  trySpendCredit(_creditNotePool, ZK_CREDIT_TYPE_UPDATE, _creditNoteProof);
  store.updateSmartAsset(_tokenId, _imprint, _interfaceProvider);
}
```

Snippet 4.8: Definition of the `updateSmartAsset` function.

Impact If a brand or a small number of brands could be linked to a single identity provider, it is possible that information about them could be leaked even if they used multiple commitments.

Recommendation We would recommend that brands only specify an interface provider if they could provide sufficient differential privacy (i.e. a sufficient number of brands would need to use the provider such that the volume of requests from the new brand would make it difficult to guess which tokens belonged to this new brand).

Developer Response We acknowledge the recommendation and agree that brands should only specify an interface provider if they can ensure sufficient differential privacy. We will also issue guidelines on this.

4.1.9 V-ARIA-VUL-009: wordPos max range is unchecked

Severity	Warning	**Commit**	7b6e47a
Type	Data Validation	**Status**	Fixed
File(s)	`UnorderedNonce.sol`		
Location(s)	invalidateUnorderedNonces		
Confirmed Fix At	c43afd0		

The `invalidateUnorderedNonces` does not check if the provided `wordPos` is less than or equal to its max value, `type(uint248).max`, and neither does this function's caller, `ArianeeIssuerProxy`.

```
// VERIDISE: UnorderedNonce.sol
function invalidateUnorderedNonces(uint256 tokenId, uint256 wordPos, uint256 mask)
    internal {
  nonceBitmap[tokenId][wordPos] |= mask;

  emit UnorderedNonceInvalidation(tokenId, wordPos, mask);
}

// VERIDISE: ArianeeIssuerProxy.sol
function invalidateUnorderedNonces(
  OwnershipProof calldata _ownershipProof,
  uint256 _tokenId,
  uint256 _wordPos,
  uint256 _mask
) external onlyWithProof(_ownershipProof, false, _tokenId) { // @audit-ok
  invalidateUnorderedNonces(_tokenId, _wordPos, _mask);
}
```

Snippet 4.9: Snippet of `invalidateUnorderedNonces` invocations.

Impact It is possible for a caller of `ArianeeIssuerProxy.invalidateUnorderedNonces` to invalidate a range of nonces that cannot be used by `_useUnorderedNonce`, which is a waste of gas.

Recommendation Add a range check in `UnorderedNonce.invalidateUnorderedNonces` that requires `wordPos` to be less than or equal to `type(uint248).max`.

Developer Response The recommended `require` statement has been added.

4.1.10 V-ARIA-VUL-010: creditNotePools cannot be removed

Severity	Warning	**Commit**	7b6e47a
Type	Access Control	**Status**	Intended Behavior
File(s)	`ArieaneeIssuerProxy.sol`		
Location(s)	N/A		
Confirmed Fix At	N/A		

A new credit note pool can be whitelisted via the `addCreditNotePool` function, but there is no method for removing a credit note pool from the `creditNotePools` whitelist.

Impact If a credit note pool becomes compromised or is updated and needs to be replaced, it cannot be, as all added pools will always be whitelisted.

Recommendation Add a `removeCreditNotePool` function that removes a credit note pool from the whitelist.

Developer Response The developers have acknowledged the issue. However, this behavior was implemented as a deliberate choice to ensure that buyers of credit notes can use them without fear that the proxy contract may be able to revoke their purchase by un-whitelisting the credit note pool.

4.1.11 V-ARIA-VUL-011: Unchecked approve return value

Severity	Warning	**Commit**	7b6e47a
Type	Data Validation	**Status**	Fixed
File(s)	`ArianeeCreditNotePool.sol`		
Location(s)	purchase		
Confirmed Fix At	4d0bb22		

The `purchase` function calls `IERC20.approve`, which returns a `bool` value indicating if the approval succeeded or not. However, the return value is ignored in the `purchase` function.

```
function purchase(CreditRegistrationProof calldata _creditRegistrationProof, bytes32
    _commitmentHash, uint256 _zkCreditType) external nonReentrant {
    // VERIDISE: ...

    // The caller should have approved the contract to transfer the amount of tokens
    token.safeTransferFrom(_msgSender(), address(this), amount);

    // Approve the store to transfer the required amount of tokens
    token.approve(address(store), amount);
    // Buy the credits from the store
    store.buyCredit(creditType, MAX_NULLIFIER_PER_COMMITMENT, issuerProxy);

    emit Purchased(creditType, _commitmentHash, insertedIndex, block.timestamp);
}
```

Snippet 4.10: Snippet from `purchase()`.

The implementation of the `purchase` function assumes that, if the `token.safeTransferFrom` call does not revert, then there should be a high enough added balance for `token.approve` to succeed. However, it is possible for the `IERC20` implementation of `token` to transfer fewer than `amount` tokens, e.g., if `token` implements a transfer tax.

Impact Not requiring `token.approve` to succeed can result in unexpected transaction reversions in `store.buyCredit` when the `ArianeeCreditNotePool` has insufficient funds to purchase the credit.

Recommendation Check if `token.approve` succeeds and revert the transaction with a meaningful error message if it fails.

Developer Response The developers have added a `require` statement for the return value of `token.approve`.

4.1.12 V-ARIA-VUL-012: Centralization Risk

Severity	Warning	**Commit**	7b6e47a
Type	Centralization	**Status**	Acknowledged
File(s)	`ArianeeIssuerProxy.sol`		
Location(s)	N/A		
Confirmed Fix At	N/A		

Similar to many projects, Arianee's, `ArianeeIssuerProxy` declares certain roles that are given special permissions. In particular, these the following roles are defined and they can take the following actions:

- Issuer
  - Can recover tokens from users (with restrictions)
  - Can destroy tokens (disabled)
  - Can create, update or destroy token events
  - Can create token messages
  - Can set token statuses as missing or stolen
  - Can update token URIs
  - Can update the token
- Owner
  - Can change commitments with the permission of the issuer
  - Can add a whitelisted `CreditNotePool`
  - Can add or remove a free credit spender

Impact If a private key were stolen, a hacker would have access to sensitive functionality that could compromise the project.

Recommendation As these are all particularly sensitive operations, we would encourage the developers to utilize a decentralized governance or multi-sig contract as opposed to a single account, which introduces a single point of failure. We would also encourage issuers to protect secrets linked to commitments using a distributed method as well. This could be done by using multiple random secrets that are encrypted with different private keys and stored in different locations.

Developer Response We will use a multi-sig contract to manage the contracts as Owner. Additionally, we will provide comprehensive guidelines for Issuers to ensure they store their secrets in a secure way.

4.1.13 V-ARIA-VUL-013: Missing address zero-checks

Severity	Warning	**Commit**	7b6e47a
Type	Data Validation	**Status**	Acknowledged
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

Description The following functions take addresses as arguments, but do not validate that the addresses are non-zero:

- ArianeeCreditNotePool.sol
  - `constructor()`: none of the `address` arguments are checked.
- ArianeeIssuerProxy.sol
  - `constructor()`: none of the `address` arguments are checked.
- MerkleTreeWithHistory.sol
  - `constructor()`: `_hasher` is not checked.

Impact If zero is passed as the address, the contract may be unable to function correctly, wasting gas due to the incorrect deployment.

Developer Response The developers have acknowledged this issue, but do not plan to make any changes to address it.

4.1.14 V-ARIA-VUL-014: Token ID reuse can have undesired side-effects

Severity	Warning	**Commit**	7b6e47a
Type	Usability Issue	**Status**	Acknowledged
File(s)	`ArianeeIssuerProxy.sol`		
Location(s)	destroy		
Confirmed Fix At	N/A		

The Arianee Proxy allows their private issuers to submit requests to destroy a token with the given ID. As seen below, the destruction process has a side-effect of un-allocating the given token ID for later reuse. Doing so can have unintended side-effects as external services may use the ownership of a token with a given ID to guard certain features.

```
function destroy(
  OwnershipProof calldata _ownershipProof,
  uint256 _tokenId
) external onlyWithProof(_ownershipProof, false, _tokenId) {
  smartAsset.destroy(_tokenId);
  // Free the commitment hash when destroying the token to allow it to be reused
  tryUnregisterCommitment(_tokenId);
}
```

Snippet 4.11: Definition of the `destroy` function.

Impact If token ownership is used externally to guard access to some functionality, an issuer (which can be anyone as pointed out by V-ARIA-VUL-005) can re-create a token with the given ID to gain access to this functionality.

Recommendation It should be noted that the `ArianeeStore` currently disallows token destruction. We would recommend that this feature not be enabled.

Developer Response We have no plans to activate this feature in the near future, but if the time comes, we will take extra care to ensure all external services relying on it are in sync.

4.1.15 V-ARIA-VUL-015: OwnershipProofs may be replayed across tokens

Severity	Warning	**Commit**	7b6e47a
Type	Replay Attack	**Status**	Fixed
File(s)	`ArianeeIssuerProxy.sol`		
Location(s)	_verifyProof		
Confirmed Fix At	f05cf98		

The Arianee Privacy module uses an `OwnershipProof` as a method of proving that the individual knows a given commitment's secrets. It also ensures that the owner approves of the action being taken by including an intent hash and that the proof cannot be replayed by including a nonce. This nonce, however, is not associated with a commitment, but rather it is associated with a token ID as shown below. As such, an `OwnershipProof` could be replayed to perform a previous action on a different token as the nonce may not be taken.

```
function _verifyProof(OwnershipProof calldata _ownershipProof, bool
    needsCreditNoteProof, uint256 _tokenId) internal {
  // VERIDISE: ...

  // Removing the ‘OwnershipProof‘ (352 bytes) and if needed the ‘CreditNoteProof‘
    (352 bytes) from the msg.data before computing the hash to compare
  uint256 msgDataHash = uint256(poseidon.poseidon([keccak256(abi.encodePacked(bytes.
    concat(msgData.slice(0, SELECTOR_SIZE), msgData.slice(SELECTOR_SIZE +
    OWNERSHIP_PROOF_SIZE + (needsCreditNoteProof ? CREDIT_NOTE_PROOF_SIZE : 0),
    msgData.length))))]));
  require(
    pIntentHash == msgDataHash,
    'ArianeePrivacyProxy: Proof intent does not match the function call'
  );

  uint256 pNonce = _ownershipProof._pubSignals[2];
  require(_useUnorderedNonce(_tokenId, pNonce), 'ArianeePrivacyProxy: Proof nonce has
     already been used');

  // VERIDISE: ...
}
```

Snippet 4.12: Snippet from `_verifyProof`.

Impact Currently this attack is infeasible due to the intent hash described above; the `pIntentHash` in the proof corresponds to a hash of the selector and arguments of the invoked function, as shown above. Therefore, for this attack to be feasible a function's arguments must not be tied to a given token's ID. Since all external functions that accept an `OwnershipProof` either take a token ID or a value implicitly associated with a single token ID as an argument, one cannot replay `OwnershipProofs`. However, this design is non-obvious and could allow replay attacks in the future.

Recommendation Rather than associating a nonce with a token ID, instead associate it with the commitment.

Developer Response Nonces are now associated with commitment hashes and not token IDs.

4.1.16 V-ARIA-VUL-016: Unused Code and Dependencies

Severity	Warning	**Commit**	7b6e47a
Type	Dead Code	**Status**	Fixed
File(s)	See issue description		
Location(s)	N/A		
Confirmed Fix At	b1dd4d7		

Description The following program constructs are unused:

- ArianeeCreditNotePool.sol
  - `ArianeeCreditNotePool is Ownable` but uses no `Ownable` features.
- ArianeeIssuerProxy.sol
  - `ArianeeIssuerProxy` is a `ReentrancyGuard` but does not use any ReentrancyGuard features.
  - The `isDefaultCreditNoteProof` function is never used.
- UnorderedNonce.sol
  - import `"@opengsn/contracts/src/ERC2771Recipient.sol"` is unused.

Impact These constructs may become out of sync with the rest of the project, leading to errors if used in the future.

Developer Response The developers have removed the unused program constructs.

4.1.17 V-ARIA-VUL-017: Typos and incorrect comments

Severity	Info	**Commit**	7b6e47a
Type	Maintainability	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	2e55c90		

Description In the following locations, the auditors identified minor typos and potentially misleading comments:

- ArianeeCreditNotePool.sol
  - `issuerProxy`: The `@notice` comment states that the `ArianeeIssuerProxy` is the only contract allowed to interact with the `ArianeeCreditNotePool` contract, but non-issuers can invoke the `purchase` method directly.
- UnorderedNonce.sol
  - `nonceBitmap`: `@notice` comment states that this field is `Used to set bits in the bitmap to prevent against signature replay protection`; should be reworded as `Used to set bits in the bitmap to prevent against signature replay attacks.` or similar.
  - `_useUnorderedNonce`: missing a `@return` doc comment like the other functions in the file.

Impact These minor errors may lead to future developer confusion.

Developer Response The developers have addressed the above issues.

Glossary

Merkle Tree A cryptographic commitment to a list of values which can be opened at individual entries in the list. See https://en.wikipedia.org/wiki/Merkle_tree to learn more. 1

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

zero-knowledge circuit A cryptographic construct that allows a prover to demonstrate to a verifier that a certain statement is true, without revealing any specific information about the statement itself. See https://en.wikipedia.org/wiki/Zero-knowledge_proof for more. 1, 29

ZK Circuit zero-knowledge circuit. 1