Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Aztec

Aztec Governance

Veridise Inc.

Oct. 13, 2025

► **Prepared For:**

Aztec Labs
https://aztec.network/

► **Prepared By:**

Tyler Diamond
Aayushman Thapa Magar
Evgeniy Shishkin

► **Contact Us:**

contact@veridise.com

► **Version History:**

Oct. 13, 2025	V2
Sep. 10, 2025	V1
Sep. 09, 2025	Initial Draft

Contents

1 Executive Summary

From Aug. 25, 2025, to Sep. 04, 2025, Aztec Labs engaged Veridise to conduct a security assessment of their Aztec Governance. The security assessment covered the smart contracts involved in the governance system utilized by the Aztec Network. Veridise conducted the assessment over 27 person-days, with 3 security analysts reviewing the project over 9 days on commit `a6eebac`. The review strategy involved a tool-assisted analysis of the program source code as well as thorough code review.

Project Summary. Aztec Network is a set of L2 blockchains represented by a collection of rollup nodes. Among these, only a single rollup node is designated as the canonical Aztec blockchain rollup, while the others operate in parallel. When Aztec protocol governance decides to introduce new functionality into the canonical rollup, the updated implementation is deployed and promoted to canonical status, while the previous implementation may continue to run.

To make upgrade operations manageable, Aztec introduced the Governance subsystem, which provides the following capabilities:

- Allows validators of different rollups to propose operations to Governance.
- Allows validators of different rollups to vote on Governance proposals.
- Allows validators of the canonical rollup to decide which proposals are worth submitting to Governance.
- Allows validators of different rollups to delegate their voting power to other users who can vote on their behalf.
- Allows validators to increase or decrease their stake in a specific rollup by depositing or withdrawing the appropriate amount of funds.
- Ensures that validator stakes follow the most recent canonical rollup, so when the canonical rollup changes, their stake is automatically reassigned to the new canonical instance.

In addition, the Governance subsystem implements the following functions:

- Provides access to the full history of rollups that have existed in Aztec.
- Provides a single authoritative answer as to which rollup is considered canonical.

The functionality is implemented through the following components:

- **Registry**: Maintains all rollup instances and serves as the source of truth for identifying the canonical chain.
- **Governance**: Holds ownership of other system components, such as the `Registry` and `CoinIssuer`. Its primary role is to implement the voting and execution logic for proposals submitted by the `GovernanceProposer` component. All participants with voting power in Governance are eligible to vote on proposals.
- **GovernanceProposer**: The only contract authorized to introduce proposals for consideration in `Governance`. It also implements a preliminary voting mechanism that pre-filters proposals based on the support of canonical rollup validators.

- **Governance Stake Escrow (GSE)**: Manages the automatic migration of validator deposits from a previous canonical chain to a new one for stakes flagged accordingly. It tracks deposited tokens of all validators across rollup instances and performs deposit and withdrawal operations on their behalf within `Governance`.

Proposal processing follows a two-step procedure. First, the `GovernanceProposer` requires a quorum of validators to signal support for a proposal within a specified number of slots*. Once the `GovernanceProposer` submits the proposal to `Governance`, either rollups collectively or validators delegates may cast their votes. After the designated voting period concludes, if the proposal has obtained sufficient support, any participant may trigger the execution of the approved proposal.

Primarily, only the `GovernanceProposer` has the right to propose to the `Governance`. In order to handle emergencies, a large holder of the underlying token can directly submit a proposal via `GSE.proposeWithLock()`.

This process allows upgrading the Layer-2 chain to a new implementation only when a significant amount of support wishes to do so. A hypothetical `Proposal` will call the `addRollup()` function in both the `Registry` and `GSE`. Contracts in the Aztec Labs ecosystem, such as 3rd party bridges, will read the `Registry` contract to determine what is considered the canonical chain.

Code Assessment. The Aztec Governance developers provided the source code of the contracts for the code review.

The source code appears to be mostly original written by the developers. It contains extensive documentation in the form of READMEs and documentation comments on functions and storage variables. To facilitate the Veridise security analysts' understanding of the code, Aztec Labs developers provided documents describing both a high-level summary of the Aztec Governance along with a deep-dive of the individual components and how they interact with one another.

The source code included a test suite with comprehensive static and fuzzing tests. Several files in the source code also indicate that the developers use linting and static analysis tools such as Solhint.

Summary of Issues Detected. The security assessment uncovered 10 issues, 1 of which was classified as medium-severity. Specifically, (V-AZGV-VUL-001) can prevent a validator from depositing into multiple rollup instances and from rejoining after leaving a rollup's validators set, which contradicts the protocol specification. Veridise analysts also identified 2 low-severity issues and, 7 warnings, including (V-AZGV-VUL-002), where if the governance contract is whitelisted to deposit, it could inflate the total voting power by transferring tokens to itself.

Recommendations. After conducting the assessment of the protocol, the security analysts had a few suggestions to improve the Aztec Governance.

Standardize Terminology. For example, the source code refers to the same entity using different terms - *validators*, *attesters*, *signalers* and *proposers*. While it is understood that different contexts

* A slot is a time interval in which a specific validator may support a proposal.

may require different labels, it should be made explicit that they all represent the same underlying entity. Clarifying this would significantly improve code readability and make comprehension easier for external reviewers.

Layer 1 Submission Assumptions. Signaling in the `GovernanceProposer` relies on the block producer (or a 3rd party using his signature) to submit his signal during his given slot. Discussion indicates that this will be 36 seconds, or 3 layer 1 slots. This means that a block producer's transaction must be included within 3 blocks, even during period of congestion. This also seems to apply to general block submission as well. Document these assumptions, and explore what a safe margin is for this value with respect to possible peaks on the layer 1.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Aztec Governance	`a6eebac`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Aug. 25–Sep. 04, 2025	Manual & Tools	3	27 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	0	0	0
Medium-Severity Issues	1	1	1
Low-Severity Issues	2	2	1
Warning-Severity Issues	7	7	7
Informational-Severity Issues	0	0	0
TOTAL	10	10	9

Table 2.4: Category Breakdown.

Name	Number
Data Validation	3
Logic Error	2
Usability Issue	2
Cryptographic Vulnerability	2
Maintainability	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Methodology

The Security Assessment process consists of the following steps:

1. Gather an initial understanding of the protocol's business logic, users, and workflows by reviewing the provided documentation and consulting with the developers.
2. Identify all valuable assets in the protocol.
3. Identify the main workflows for managing these assets.
4. Identify the most significant security risks associated with these assets.
5. Systematically review the codebase for execution paths that could trigger the identified security risks, considering different assumptions.
6. Prioritize one finding over another by assigning a severity level to each.

The Veridise security analysts perused two documents shared at the beginning of the project:

- A Quick Tour of Governance
- Aztec Governance description

3.2 Identified Security Risks

After the initial phase of the security assessment was completed, a list of potential security risks was generated. Security analysts used this list during the code review as a starting point to identify potential attack vectors. A few of these risks, when expressed as questions, include the following:

- **Deposits**
  - Is there a way to steal or lock validator deposits?
  - Is there a way to prevent legitimate users from depositing?
  - Is the fund withdrawal logic implemented as described?
  - Are there any centralization risks in the funds management logic?
- **Voting**
  - Is there a way to subvert voting thresholds?
  - Can validators vote with more power than they hold at a given timestamp?
  - Can validators increase their voting power without depositing the required funds, or decrease others' voting power?
  - Can a malicious validator subvert the voting of other validators?
  - Is it possible to steal voting power by reassigning a delegatee to a validator?
  - Is it possible to subvert voting results?
- **Proposals**
  - Is there a way to bypass proposal origin restrictions, such as whitelisting?
  - Are proposal time delays correctly implemented in the relevant logic?

  - Can proposals hide any of the code that will be executed?
  - What guarantees exist for submitting signals to the `GovernanceProposer`?
- **Rewards Distribution**
  - Can rewards designated for a canonical rollup be claimed by other rollups?
  - Can the rewards claiming process be subverted or manipulated?
- **Governance**
  - Is it possible for an attacker to take over governance or to submit and rapidly execute a malicious proposal?
- **General Considerations**
  - Is the asset coin issuance mechanism sound?
  - Are all contracts within the project in sync?
  - Are there any front-running risks in the main protocol workflows?
  - Are there any code-level vulnerabilities, such as arithmetic overflows, underflows or reentrancies?
  - Are all access control mechanisms in place?
  - Are all cryptographic mechanisms implemented and composed in a secure manner?

3.3 Scope

The scope of this security assessment was limited to a specific set of source files, as agreed upon with the Aztec Governance developers:

- `l1-contracts/src/governance/CoinIssuer.sol`
- `l1-contracts/src/governance/Governance.sol`
- `l1-contracts/src/governance/GSEPayload.sol`
- `l1-contracts/src/governance/GSE.sol`
- `l1-contracts/src/governance/interfaces/ICoinIssuer.sol`
- `l1-contracts/src/governance/interfaces/IEmpire.sol`
- `l1-contracts/src/governance/interfaces/IGovernanceProposer.sol`
- `l1-contracts/src/governance/interfaces/IGovernance.sol`
- `l1-contracts/src/governance/interfaces/IPayload.sol`
- `l1-contracts/src/governance/interfaces/IProposerPayload.sol`
- `l1-contracts/src/governance/interfaces/IRegistry.sol`
- `l1-contracts/src/governance/interfaces/IRewardDistributor.sol`
- `l1-contracts/src/governance/libraries/AddressSnapshotLib.sol`
- `l1-contracts/src/governance/libraries/CheckpointedUintLib.sol`
- `l1-contracts/src/governance/libraries/ConfigurationLib.sol`
- `l1-contracts/src/governance/libraries/DepositDelegationLib.sol`
- `l1-contracts/src/governance/libraries/Errors.sol`
- `l1-contracts/src/governance/libraries/ProposalLib.sol`
- `l1-contracts/src/governance/proposer/EmpireBase.sol`
- `l1-contracts/src/governance/proposer/GovernanceProposer.sol`
- `l1-contracts/src/governance/Registry.sol`
- `l1-contracts/src/governance/RewardDistributor.sol`
- `l1-contracts/src/shared/interfaces/IMintableERC20.sol`

- `l1-contracts/src/shared/libraries/BN254Lib.sol`
- `l1-contracts/src/shared/libraries/CompressedTimeMath.sol`
- `l1-contracts/src/shared/libraries/SignatureLib.sol`
- `l1-contracts/src/shared/libraries/TimeMath.sol`

3.4 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

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

The likelihood of a vulnerability is evaluated according to the Table 3.2.

Table 3.2: Likelihood Breakdown

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

The impact of a vulnerability is evaluated according to the 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 Security Review Assumptions

4.1 Operational Assumptions

In addition to assuming that any external libraries behaved correctly and are free of security vulnerabilities, the Veridise analysts made the following assumptions for their security assessment:

- **Rollup Contracts Implementation**. The implementation of Rollup contracts is assumed to be free of vulnerabilities that could allow a malicious party to execute calls on behalf of the Rollups within the Governance protocol.
- **Rollup Voting**. It is assumed that Rollup owners actively participate in Governance voting, ensuring representation of their interests and supporting the decentralization and legitimacy of the decision-making process.
- **Proposals Evaluation**. It is assumed that validators and Rollup owners possess the technical expertise to critically evaluate governance proposals, ensuring that decisions are informed and technically sound.
- **Governance Staking Escrow (GSE) setup**. The `GSE` setup must be executed exactly as specified, in two phases: initially, ownership resides with the deployer, and subsequently, ownership is transferred to the Governance contract.
- **Registry setup**. The setup of `Registry` must be executed exactly as specified, in two phases: initially, ownership resides with the deployer, and subsequently, ownership is transferred to the Governance contract.
- **Asset Coin Minters**. The Asset Coin contract is expected to implement the `addMinter()` and `removeMinter()` functions. It was assumed that these functions are not misused, as improper use could enable uncontrolled token minting.

4.2 Operational Recommendations.

Highly-privileged operations should be executed using a multi-sig contract or decentralized governance system. These operations should be guarded by a timelock to ensure there is enough time for incident response. Highly-privileged, emergency operations should be tested in example scenarios to ensure the role operators are available and ready to respond when necessary.

Full validation of operational security practices is beyond the scope of this review. Users of the protocol should ensure they are confident that the operators of privileged keys are following best practices such as:

- Never storing a protocol key in plaintext, on a regularly used phone, laptop, or device, or relying on a custom solution for key management.
- Using separate keys for each separate function.
- Storing multi-sig keys in a diverse set of key management software/hardware services and geographic locations.

- Enabling 2FA for key management accounts. SMS should *not* be used for 2FA, nor should any account which uses SMS for 2FA. Authentication apps or hardware are preferred.
- Validating that no party has control over multiple multi-sig keys.
- Performing regularly scheduled key rotations for high-frequency operations.
- Securely storing physical, non-digital backups for critical keys.
- Actively monitoring for unexpected invocation of critical operations and/or deployed attack contracts.
- Regularly drilling responses to situations requiring emergency response such as pausing/unpausing.

5 Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 5.1 summarizes the issues discovered:

Table 5.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-AZGV-VUL-001	Public key check in deposit function breaks . . .	Medium	Fixed
V-AZGV-VUL-002	Governance deposit from self can inflate . . .	Low	Fixed
V-AZGV-VUL-003	Payload may change actions after passing . . .	Low	Acknowledged
V-AZGV-VUL-004	Maintainability issues	Warning	Fixed
V-AZGV-VUL-005	Multiple addRollup calls silently overwrite . . .	Warning	Fixed
V-AZGV-VUL-006	Missing curve order checks may lead to . . .	Warning	Fixed
V-AZGV-VUL-007	Non-deterministic algorithm of . . .	Warning	Fixed
V-AZGV-VUL-008	Missing enforcement of withdrawer . . .	Warning	Fixed
V-AZGV-VUL-009	Non-standard hash-to-curve implementation	Warning	Fixed
V-AZGV-VUL-010	Misleading function nomenclature / . . .	Warning	Fixed

5.1 Detailed Description of Issues

5.1.1 V-AZGV-VUL-001: Public key check in deposit function breaks attester reusability

Severity	Medium	**Commit**	a6eebac
Type	Logic Error	**Status**	Fixed
Location(s)	`src/governance/GSE.sol:336`		
Confirmed Fix At	`https://github.com/AztecProtocol/aztec-packages/pull/17247/`, cab5a54		

Description The documentation surrounding the `deposit()` function in the `GSE` specifies that an `attester` can deposit on multiple rollup instances. Additionally, it specifies that an `attester` will have to register a new public key if they were to fully withdraw from the contract. This is documented by the following comment:

```
/* @dev The same attester may deposit on multiple *instances*, so long as
/* the latest-rollup-instance-attesters-form-set invariant described above
    BONUS_INSTANCE_ADDRESS holds. */
...

// This is the ONLY place where we set the configuration for an attester.
// This means that their withdrawer and public keys are set once, globally.
// If they exit, they must re-deposit with a new key.
```

Snippet 5.1: Snippet from `governance/GSE.sol:deposit()`

However, `deposit()` calls `_checkProofOfPossession()`, which checks that the `attester`'s registered public key is zero. Therefore, an `attester` can never call `deposit()` again.

Impact An `attester` can only every deposit once and therefore can only register on a single instance (or the bonus instance). Additionally, after an `attester` has withdrawn, they cannot register in the `GSE` again.

Recommendation Remove the check that previously registered public key is zero when an `attester` is attempting to register to a separate instance. Additionally, If the documentation refers to requiring the `attester` to utilize a new Ethereum address, then change the wording to indicate as such.

Developers Response The developers mention that the comment relating to the described behavior is wrong, and have updated it.

5.1.2 V-AZGV-VUL-002: Governance deposit from self can inflate power

Severity	Low	**Commit**	a6eebac
Type	Data Validation	**Status**	Fixed
Location(s)	`src/governance/Governance.sol:323-328`		
Confirmed Fix At	`https://github.com/AztecProtocol/aztec-packages/pull/16917`, 95c296f		

Description The `deposit()` function in `Governance` will transfer `ASSET` tokens from the `msg.sender` to itself, increase the balance of power of the `msg.sender`, and increase the `total` power in the contract.

```
function deposit(address _beneficiary, uint256 _amount) external override(IGovernance
    ) isDepositAllowed(_beneficiary) {
    ASSET.safeTransferFrom(msg.sender, address(this), _amount);
    users[_beneficiary].add(_amount);
    total.add(_amount);

    emit Deposit(msg.sender, _beneficiary, _amount);
  }
```

Snippet 5.2: Snippet from `governance/Governance.sol:deposit()`

If `Governance` is ever whitelisted in the `depositControl.isAllowed` mapping, a proposal could, in theory, be created that invokes this function. Because proposals are executed by `Governance`, this would artificially inflate the total voting power, which in turn would raise the thresholds required for proposals to pass.

Impact The `Governance` itself will be given power to vote with that can only be used by execution of another proposal. More importantly, the thresholds required for forming a quorum and passing a proposal will be inflated. If the `_amount` is sufficiently high, or this function is called multiple times, then the Governance system may be in a state that will make it impossible to pass proposals. This also bypasses the check in `execute()` that the `ASSET` cannot be a target.

Recommendation Explicitly prohibit `address(this)` from calling this function.

Developers Response The developers added a check in the `isDepositAllowed()` modifier to ensure that the governance contract cannot deposit onto itself.

5.1.3 V-AZGV-VUL-003: Payload may change actions after passing Governance

Severity	Low	**Commit**	a6eebac
Type	Logic Error	**Status**	Acknowledged
Location(s)	`src/governance/GSEPayload.sol:62`		
Confirmed Fix At	N/A		

Description The `GSEPayload` constructor accepts a contract that implements the `IPayload` interface. This contract is created and submitted to Governance for voting, but only after its underlying *original* payload has satisfied the signaling requirements defined by `GovernanceProposer`.

```
1 function getActions() external view override(IPayload) returns (IPayload.Action[]
      memory) {
2     IPayload.Action[] memory originalActions = ORIGINAL.getActions();
3     IPayload.Action[] memory actions = new IPayload.Action[](originalActions.length +
       1);
4
5         // [Veridise] elided
6 }
```

Snippet 5.3: Snippet from `src/governance/GSEPayload.sol:getActions()`

The original payload contract can be deployed by anyone. As a result, it may act as a proxy or return values conditionally (for example, based on `msg.sender` or the current time). Furthermore, the `proposeWithLock()` function in `Governance` directly accepts a proposal address to be put up for a vote.

Impact If the signalers in `GovernanceProposer` and the voters in `Governance` do not carefully review the underlying *original* payload, unintended actions could be executed.

Recommendation

1. For the `GSEPayload`, store the actions from `ORIGINAL.getActions()` directly in contract itself. This will prevent the actions from changing during voting in `Governance`.
2. For the `proposeWithLock()` route, take the list of actions as a parameter to the function and then deploy an immutable clone that stores them.

Developers Response Developers have acknowledged the issue and have indicated they do not plan to provide a fix. They state:

- The issue relies on actors not reviewing what they are voting on. If this is the case, they might also vote payloads that are clearly malicious as well.

5.1.4 V-AZGV-VUL-004: Maintainability issues

Severity	Warning	**Commit**	a6eebac
Type	Maintainability	**Status**	Fixed
Location(s)	`src/governance/RewardDistributor.sol`		
Confirmed Fix At	https://github.com/AztecProtocol/aztec-packages/pull/17248, a310dd9		

Description The maintainability of the code may be improved by changes in the following locations:

1. `governance/RewardDistributor.sol`:
   - The`claim()` and `recover()` functions should sanity check the destination address
2. `governance/GSE.sol`:
   - The `IGSECore` and `IGSE` interfaces should be in a separate file.
3. `governance/interfaces/IGovernance.sol`:
   - Rename the `ProposeConfiguration` to `ProposeWithLockConfiguration` to more accurately describe its usage.
4. `governance/Governance.sol`:
   - The `vote()` function only ever returns true, and callers do not check the result.
5. `governance/libraries/AddressSnapshotLib.sol`:
   - The documentation of the `remove(address)` function states that it is only used in tests, but it is used in `GSE.withdraw()`.
6. `shared/libraries/BN254.sol`:
   - `sqrt()`: Use the `BASE_FIELD_ORDER` named variable instead of a literal constant.
7. `governance/libraries/ConfigurationLib.sol`:
   - `assertValid()`: The function only ever returns true, and callers do not check the result.

The security analysts identified the following code that is unused outside of its definition:

1. `governance/libraries/AddressSnapshotLib.sol`:
   - `AddressSnapshotLib__AddressNotInSet`
2. `governance/GSE.sol`:
   - `_getInstanceStoreWithAttester()`
3. `governance/libraries/ProposalLib.sol`:
   - `VoteTabulationInfo.MinimumEqZero`

- `VoteTabulationInfo.YeaLimitEqVotesCast`

4. `governance/libraries/Errors.sol`:
   - `Governance__NoCheckpointsFound`
   - `Governance__InvalidConfiguration`
   - `Governance__ProposalLib__ZeroMinimum`
   - `Governance__ProposalLib__ZeroVotesNeeded`
   - `Governance__ProposalLib__MoreVoteThanExistNeeded`
   - `Governance__ProposalLib__ZeroYeaVotesNeeded`
   - `Governance__ProposalLib__MoreYeaVoteThanExistNeeded`
   - `GovernanceProposer__PayloadHaveNoCode`
   - `Governance__BlsKeyInvalidG1Point`
   - `Governance__BlsKeyInvalidG2Point`

Developers Response The developers addressed all the points mentioned above except for 1 and 2.

5.1.5 V-AZGV-VUL-005: Multiple addRollup calls silently overwrite the stored rollups in GSE

Severity	Warning	**Commit**	a6eebac
Type	Data Validation	**Status**	Fixed
Location(s)	`src/governance/GSE.sol:260`		
Confirmed Fix At	`https://github.com/AztecProtocol/aztec-packages/pull/17246`, 7a12a96, 46e3dd7		

Description The `addRollup()` function in `GSE` is used to add a new rollup to the stored `instances` and the `rollups` Checkpoints array. The OpenZeppelin Checkpoints structures allow one to track the state of a value across time and later lookup what the value was at a given time. It can be thought of as an append-only array that will store the `(block.timestamp, value)` tuple on every `push()` call. However, calls with the same `block.timestamp` will overwrite the entry.

If a proposal executes `addRollup()` multiple times, only the most recently added rollup will be retained, while the earlier ones will be discarded.

Impact The list of rollups that have ever been added would become inaccurate. However, security analysts note that this scenario is highly unlikely, given the intended context of the function's usage and the expected infrequency of adding new rollups.

Recommendation Document this assumption or check that the latest value's time key is not equal to the current `block.timestamp`

Developers Response Developers have updated the comments surrounding the relevant code.

5.1.6 V-AZGV-VUL-006: Missing curve order checks may lead to malleability issues

Severity	Warning	**Commit**	a6eebac
Type	Data Validation	**Status**	Fixed
Location(s)	`src/shared/libraries/BN254Lib.sol:264-339`		
Confirmed Fix At	`https://github.com/AztecProtocol/aztec-packages/pull/17249`, 5cd8f41		

Description The `isOnCurveG1()` and `isOnCurveG2()` functions check whether a given point lies on the corresponding BN254 curves. They do this by verifying that the `x` and `y` coordinates satisfy the elliptic curve equation. However, the functions do not enforce that the coordinates fall within the base field's order, which can result in multiple valid representations of the same point.

Impact Currently, these functions are only used within `proofOfPossession()`. This function also invokes Ethereum precompiles, which revert when provided with points whose coordinates fall outside the base field. However, if these functions are used in other contexts, malleability issues could arise.

Recommendation Ensure that the coordinates are less than `BASE_FIELD_ORDER`. If performing this check is too gas-expensive, explicitly document this assumption.

Developers Response The developers have removed the `isOnCurve` functions.

5.1.7 V-AZGV-VUL-007: Non-deterministic algorithm of hashToPoint can lead to gas-exhaustion attacks

Severity	Warning	**Commit**	a6eebac
Type	Usability Issue	**Status**	Fixed
Location(s)	`src/shared/libraries/BN254Lib.sol:209-227`		
Confirmed Fix At	`https://github.com/AztecProtocol/aztec-packages/pull/17249`, 5cd8f41		

The `hashToPoint()` function maps the hash of a `domain` and `message` to a point on the BN254 G1 group.

```
function hashToPoint(bytes32 domain, bytes memory message) internal view returns (
    G1Point memory output) {
  bytes32 hashed = keccak256(abi.encode(domain, message));
  uint256 x = uint256(hashed) % BASE_FIELD_ORDER;
  uint256 y;
  bool found = false;
  while (true) {
    y = mulmod(x, x, BASE_FIELD_ORDER);
    y = mulmod(y, x, BASE_FIELD_ORDER);
    y = addmod(y, 3, BASE_FIELD_ORDER);
    (y, found) = sqrt(y);
    if (found) {
      output = G1Point({x: x, y: y});
      break;
    }
    x = addmod(x, 1, BASE_FIELD_ORDER);
  }
  require(found, NoPointFound());
  return output;
}
```

Snippet 5.4: Snippet from `shared/libraries/BN254Lib.sol:hashToPoint()`

After the initial `x` value is derived from the hash of the domain and message, the algorithm checks whether a corresponding `y` value exists to form a valid point on the curve. If no such point exists, it repeatedly increments `x` until a valid `y` is found. Since roughly half of the `x` values in the field do not map to a valid `y`, the probability of success per iteration is about 1/2. As a result, this loop is not deterministic.

Impact An attacker could craft a message that requires many iterations of this loop, potentially consuming a large amount of gas. In the current context, it seems that only the attacker would bear this cost. However, if the function is ever exposed to third parties-for example, when verifying a message signature-those parties would be forced to pay the excessive gas cost.

Recommendation Explore implementing deterministic alternatives such as those defined in RFC9380 or the Fouque-Tibouchi technique. If these alternatives are too expensive and this function is not used in a dangerous manner as mentioned above, then clearly document this behavior.

Developer Response We accept the non-deterministic gas usage. But put a limit on the full gas for the proof-of-possession computation, so there will directly be a limit of what we accept. This will potentially lead to certain "good" inputs being rejected, but we find that acceptable as the gas cost of the deterministic approaches are too high in the happy case.

5.1.8 V-AZGV-VUL-008: Missing enforcement of withdrawer authorization within GSE Contract

Severity	Warning	**Commit**	a6eebac
Type	Usability Issue	**Status**	Fixed
Location(s)	`src/governance/GSE.sol:350`		
Confirmed Fix At	`https://github.com/AztecProtocol/aztec-packages/pull/17247`, cab5a54		

Description The `deposit()` function in the GSE contract accepts a `_withdrawer` argument, described as *"Address which can initiate a withdraw for the* `_attester`*"*. However, the GSE contract itself does not enforce that withdrawals can only be initiated by this `_withdrawer`. Instead, It is identified that it relies on the rollup contract's `initiateWithdraw()` implementation to enforce this restriction.

Impact Due to this design dependency, the correctness of GSE's withdrawal authorization is dependent upon the rollup contract continuing to perform the necessary checks. Should a new rollup contract be deployed without preserving this logic, the intended access control mechanism in GSE may be bypassed.

Recommendation Document that the Rollup contracts that utilize the GSE must enforce the originator of withdrawals to match this value.

Developers Response The developers updated the comments surrounding the relevant code.

5.1.9 V-AZGV-VUL-009: Non-standard hash-to-curve implementation

Severity	Warning	**Commit**	a6eebac
Type	Cryptographic Vulnerabilit	**Status**	Fixed
Location(s)	`src/shared/libraries/BN254Lib.sol`		
Confirmed Fix At	`https://github.com/AztecProtocol/aztec-packages/pull/17262`, 1856c61		

Description The used signature scheme requires a hash function (modeled as a random oracle) into the group G1. This is achieved by hashing to a field element, which constitutes a potential x-coordinate of the point in G1, and incrementing it until a suitable y-coordinate is found (until x^3+3 is a square in the base field Fp). The implementation deviates from the description in Section 3.3 of "Short signatures from the Weil pairing" in several ways, which renders the corresponding security argument inapplicable to this case:

1. **Incrementing H(message) does not agree with counter-in-the-hash** In standard security arguments each of the consecutive trials is modeled as a fresh independent random oracle call. Having a counter-in-the-hash (H(message || counter)) achieves this. Post-processing one digest (e.g. having a single hash and incrementing it as is used in the implementation) does not. The consecutive trials are dependent.
2. **Choice of sign of square-root independent of message**
   For a valid x-coordinate, the sign of the square-root of x^3+3 is chosen independent of the message (as the (p+1)/4-th power of x^3+3 modulo p). Hence only half of the points on the elliptic curve are in the range of the hash map. The choice of the square-root (its sign) should be determined as an output of the hash function, which should output an element from Fp × 0,1.

Impact Security proofs for BLS signatures model the hash to curve as a random oracle. With the mentioned deviations, this is not given anymore and hence the setup is not covered by theoretical soundness proofs. Hence one cannot claim provably security in the random oracle model anymore.

Recommendation Use the Shallue–van de Woestijne method for hashing into G1 or implement the try-and-increment method as described in Section 3.3 [here] with the counter-in-hash (rather than postprocessing one digest) and choice of sign of y according to the final bit output of the hash function.

Developers Response The developers now take the hash of `(domain, message, counter)` for every attempt of computing valid coordinates. Additionally, once a valid pair of coordinates is found, the parity of `y` is now based on the least singificant bit of the hash of `(domain, message, counter+1)`.

5.1.10 V-AZGV-VUL-010: Misleading function nomenclature / missing subgroup check

Severity	Warning	**Commit**	a6eebac
Type	Cryptographic Vulnerabilit	**Status**	Fixed
Location(s)	`src/shared/libraries/BN254Lib.sol`		
Confirmed Fix At	`https://github.com/AztecProtocol/aztec-packages/pull/17249`, 5cd8f41		

Description The subgroup G2 of the group of Fp^2-rational points on the twist E′ of the BN254 curve has index not equal to 1, i.e. G2 is a proper subset of E′(Fp^2). The function `isOnCurveG2` only checks if the given point coordinates satisfy the equation of E′, but not if the point is in the prime order subgroup. The naming does not correspond to the implemented functionality of the function.

Impact The behavior of the pairing operation on points not in the prime order subgroups is undefined, which can be exploited to enable forgeries (see Section 5.3 here). Membership check for the prime order subgroup is required for the verification of BLS signatures. The BN254 pairing precompile checks for G2 membership. Hence this should not pose a risk. However because of the nomenclature a user of the `isOnCurveG2` function might erroneously assume that the check is performed, which might result in unexpected behavior and vulnerabilities.

Recommendation Change the name of the function to reflect the checks that are performed or add a subgroup check.

Developers Response The developers have removed the `isOnCurve` functions.