



Ayni

Security Assessment

CertiK Assessed on Oct 15th, 2025





CertiK Assessed on Oct 15th, 2025

Ayni

The security assessment was prepared by CertiK.

Executive Summary

TYPES

Staking

ECOSYSTEM

EVM Compatible

METHODS

Manual Review, Static Analysis

LANGUAGE

Solidity

TIMELINE

Preliminary comments published on 10/07/2025

Final report published on 10/15/2025

Vulnerability Summary



12

Total Findings

10

Resolved

0

Partially Resolved

2

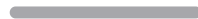
Acknowledged

0

Declined

2 Centralization

2 Acknowledged



Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.

0 Critical

Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.

0 Major

Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.

1 Medium

1 Resolved



Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.

5 Minor

5 Resolved



Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.

4 Informational

4 Resolved



Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | AYNi

Summary

[Executive Summary](#)

[Vulnerability Summary](#)

[Codebase](#)

[Audit Scope](#)

[Approach & Methods](#)

Findings

[AYN-04 : Centralization Risks In AyniStaking.Sol](#)

[AYN-05 : Over-Reliance On Backend Operations Obscures Transparency](#)

[AYN-06 : Vulnerable Claim State Management Allows Out-Of-Order Claims Requiring Manual Intervention](#)

[AYN-07 : Incomplete Validation In `stakes` Check Allows Reuse](#)

[AYN-08 : Missing `endTime` Validation In Input Check](#)

[AYN-09 : Redundant Nonce Tracking With Overly Specific Parameters In `claim\(\)` Function](#)

[AYN-10 : Inconsistent Input Validation Between `addSigner\(\)` And `removeSigner\(\)` Functions](#)

[AYN-11 : Unsafe Fee Collection Pattern](#)

[AYN-12 : Missing Emit Events](#)

[AYN-13 : Obsolete And Inaccurate Comments](#)

[AYN-14 : Inconsistent Parameter Order Between Encoding And Signature Verification In `claim\(\)` Function](#)

[AYN-15 : Potentially Misleading Event Parameter Name `lastPreclaimMonth` Contains Unclear "Pre" Prefix](#)

Optimizations

[AYN-02 : `NotMatured` error is never used](#)

Appendix

Disclaimer

CODEBASE | AYNİ

Repository

<https://github.com/pixelette-technologies/ayni-staking-contract/tree/393dc92768ac8cbd88ee45807ec45856801daa2d/contracts>

<https://github.com/pixelette-technologies/ayni-staking-contract/tree/44c05a2dae58af3e416fe7287215079c0e51a0c1/contracts>

Commit

[393dc92768ac8cbd88ee45807ec45856801daa2d](https://github.com/pixelette-technologies/ayni-staking-contract/tree/393dc92768ac8cbd88ee45807ec45856801daa2d)

[44c05a2dae58af3e416fe7287215079c0e51a0c1](https://github.com/pixelette-technologies/ayni-staking-contract/tree/44c05a2dae58af3e416fe7287215079c0e51a0c1)

AUDIT SCOPE | AYNI

pixelette-technologies/ayni-staking-contract

 AyniStaking.sol

APPROACH & METHODS | AYNI

This audit was conducted for Ayni to evaluate the security and correctness of the smart contracts associated with the Ayni project. The assessment included a comprehensive review of the in-scope smart contracts. The audit was performed using a combination of Manual Review and Static Analysis.

The review process emphasized the following areas:

- Architecture review and threat modeling to understand systemic risks and identify design-level flaws.
- Identification of vulnerabilities through both common and edge-case attack vectors.
- Manual verification of contract logic to ensure alignment with intended design and business requirements.
- Dynamic testing to validate runtime behavior and assess execution risks.
- Assessment of code quality and maintainability, including adherence to current best practices and industry standards.

The audit resulted in findings categorized across multiple severity levels, from informational to critical. To enhance the project's security and long-term robustness, we recommend addressing the identified issues and considering the following general improvements:

- Improve code readability and maintainability by adopting a clean architectural pattern and modular design.
- Strengthen testing coverage, including unit and integration tests for key functionalities and edge cases.
- Maintain meaningful inline comments and documentations.
- Implement clear and transparent documentation for privileged roles and sensitive protocol operations.
- Regularly review and simulate contract behavior against newly emerging attack vectors.

FINDINGS | AYNi



12

Total Findings

0

Critical

2

Centralization

0

Major

1

Medium

5

Minor

4

Informational

This report has been prepared for Ayni to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the audit, a total of 12 issues were identified. Leveraging a combination of Manual Review & Static Analysis the following findings were uncovered:

ID	Title	Category	Severity	Status
AYN-04	Centralization Risks In AyniStaking.Sol	Centralization	Centralization	● Acknowledged
AYN-05	Over-Reliance On Backend Operations Obscures Transparency	Centralization	Centralization	● Acknowledged
AYN-06	Vulnerable Claim State Management Allows Out-Of-Order Claims Requiring Manual Intervention	Logical Issue	Medium	● Resolved
AYN-07	Incomplete Validation In <code>stakes</code> Check Allows Reuse	Logical Issue	Minor	● Resolved
AYN-08	Missing <code>endTime</code> Validation In Input Check	Inconsistency	Minor	● Resolved
AYN-09	Redundant Nonce Tracking With Overly Specific Parameters In <code>claim()</code> Function	Logical Issue	Minor	● Resolved
AYN-10	Inconsistent Input Validation Between <code>addSigner()</code> And <code>removeSigner()</code> Functions	Inconsistency	Minor	● Resolved
AYN-11	Unsafe Fee Collection Pattern	Logical Issue	Minor	● Resolved
AYN-12	Missing Emit Events	Coding Style	Informational	● Resolved
AYN-13	Obsolete And Inaccurate Comments	Coding Style	Informational	● Resolved

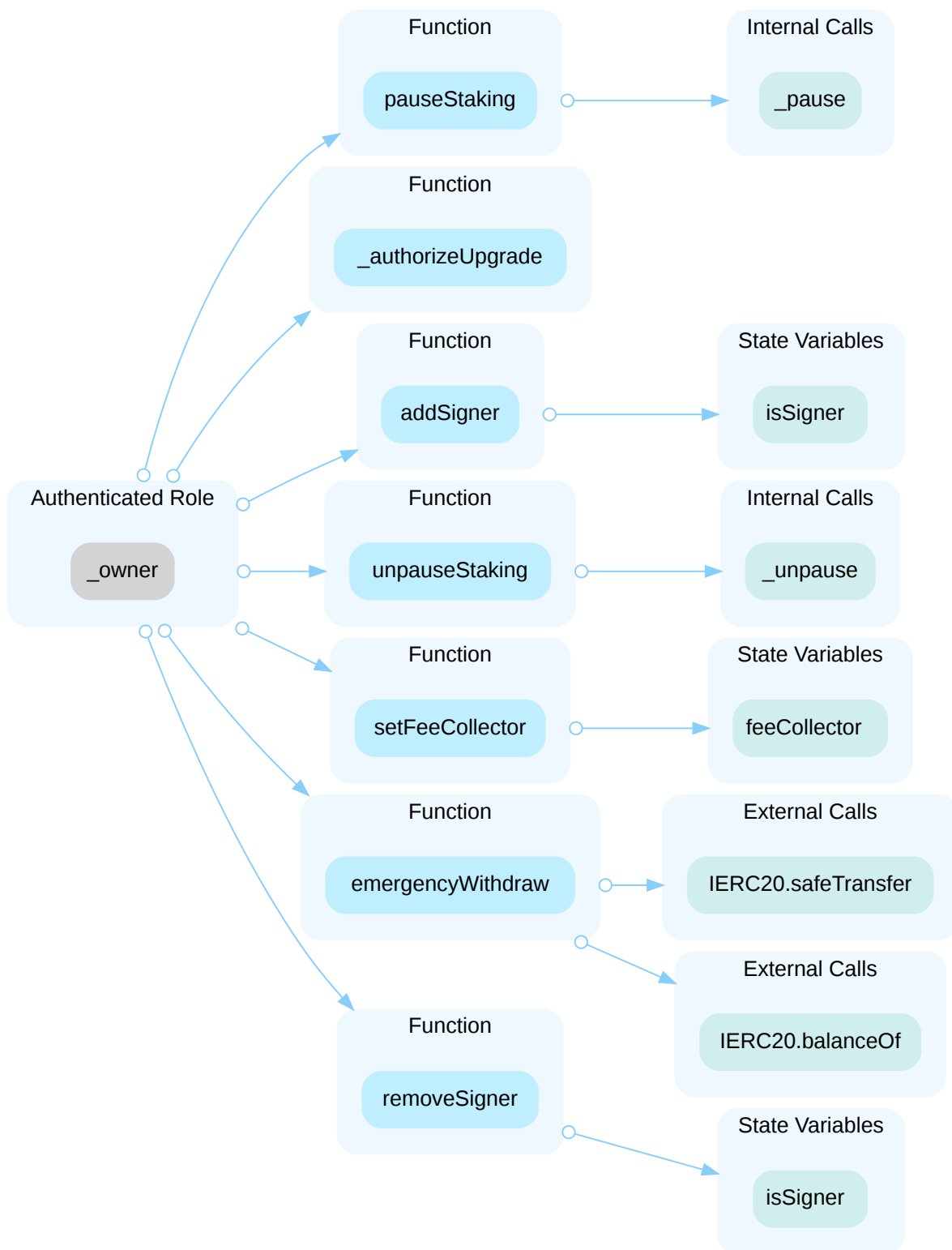
ID	Title	Category	Severity	Status
AYN-14	Inconsistent Parameter Order Between Encoding And Signature Verification In <code>claim()</code> Function	Inconsistency	Informational	● Resolved
AYN-15	Potentially Misleading Event Parameter Name <code>lastPreclaimMonth</code> Contains Unclear "Pre" Prefix	Coding Style	Informational	● Resolved

AYN-04 | Centralization Risks In AyniStaking.Sol

Category	Severity	Location	Status
Centralization	● Centralization	AyniStaking.sol: 455, 465, 478, 490, 507, 514, 521	● Acknowledged

Description

In the contract `AyniStaking` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and assign a signer to withdraw all the funds.



Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts

with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign ($\frac{2}{3}$, $\frac{3}{5}$) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
OR
- Remove the risky functionality.

I Alleviation

[Ayni, 10/15/2025]: This behavior is intentional and designed to ensure that the owner retains authority over all functions outlined in the diagram. Ownership will be managed securely through a multisig wallet.

AYN-05 | Over-Reliance On Backend Operations Obscures Transparency

Category	Severity	Location	Status
Centralization	● Centralization	AyniStaking.sol (base): 330	● Acknowledged

Description

The provided code demonstrates a significant over-reliance on backend operations. Not only are funds transferred based on backend-signed messages, but the calculation of staking rewards is also performed by the backend. This approach poses multiple issues.

In general, the primary appeal of using smart contracts is their inherent transparency and immutability. However, as seen here, the smart contract essentially serves as a wrapper over backend logic, rather than standing alone as an independent, fully-transparent, and verifiable source of truth.

This excessive dependence on the backend reduces the contract's transparency and increases the probability of mistakes or unwanted scenarios as the backend logic is opaque to users. Moreover, it leaves the users out of control over their funds, as these operations could potentially be performed without their explicit knowledge and permission.

Recommendation

It is highly recommended to reduce the dependence on backend operations and shift to a more transparent and user-controlled model. By doing so, users can directly interact with the contract, not only to stake their own funds but also to calculate and claim their rewards. This would enhance the transparency and verifiability of the contract while also empowering the users by giving them more control.

Alleviation

[Ayni, 10/15/2025]: This is intentional and is required by the project design overall.

AYN-06 | Vulnerable Claim State Management Allows Out-Of-Order Claims Requiring Manual Intervention

Category	Severity	Location	Status
Logical Issue	● Medium	AyniStaking.sol (base): 428-434	● Resolved

Description

The claim logic contains two state management issues:

- 1. Out-of-order claim vulnerability:** The `claimedUntilMonth` is directly overwritten without validation:

```
userStake.claimedUntilMonth = claimedMonth;
```

This allows users to submit claims in any order, potentially rolling back the claimed period if an earlier month is submitted after a later month has been claimed. While the backend is trusted and won't sign malicious claims, users can strategically submit signed claims out of order.

- 2. Premature stake state transition:** The stake can enter an inconsistent state:

```
if (claimedMonth >= interval && block.timestamp >= userStake.endTime) {
    userStake.isClaimed = true;
    userStake.isActive = false;
    // ... return staking tokens
}
```

If `claimedMonth >= interval` but `block.timestamp < userStake.endTime`, the user cannot receive the staking tokens back. While not permanently stuck, getting the staked funds requires the backend to sign and submit a "dummy" claim message with zero `rewards` to be submitted after `userStake.endTime`.

Recommendation

- 1. Add claim order validation:** Ensure `claimedMonth > userStake.claimedUntilMonth` before updating the claimed period to prevent out-of-order submissions
- 2. Fix stake finalization logic:** Consider separating the "final claim" from "stake withdrawal" into distinct operations to avoid intermediate inconsistent states

Alleviation

[Ayni, 10/15/2025]: In our implementation, the claim flow is fully backend-driven with the following safeguards that eliminate practical risk of out-of-order submissions:

Short-lived signatures: Each claim signature includes an expiry field, typically valid for less than two minutes. This prevents replaying older signed claims later.

Strict backend sequencing: The backend only signs claim requests when a claim period becomes due (after, every 3 months) and never reissues signatures for previous periods if already claimed.

One-time salt and nonce: Each signature includes a unique salt and nonce combination that cannot be reused on-chain.

Given these measures, it's not possible for a user to resubmit an older claim after a later one — the signature would be expired.

The `endTime` enforces the minimum staking duration, while `interval` represents the total staking period.

The condition `if (claimedMonth >= interval && block.timestamp >= userStake.endTime)` ensures that if `claimedMonth` is less than the full interval, the staked tokens remain locked until `block.timestamp >= endTime`.

Therefore, early reward claiming does not enable early unstaking.

AYN-07 | Incomplete Validation In `stakes` Check Allows Reuse

Category	Severity	Location	Status
Logical Issue	● Minor	AyniStaking.sol (base): 187, 313	● Resolved

Description

```
if (stakes[userId][interval][stakeId].isActive) revert AlreadyStaked();
```

The provided code performs a check on whether a particular stake associated with a `userId` and `interval` is active. However, this check doesn't consider the case where a stake is not active but is already claimed. This oversight can potentially allow for an existing stake to be reused, which can lead to unintended behavior or potential security risks.

Recommendation

Extend the verification to include whether the stake is already claimed.

AYN-08 | Missing `endTime` Validation In Input Check

Category	Severity	Location	Status
Inconsistency	● Minor	AyniStaking.sol (base): 287	● Resolved

Description

The provided code checks several input conditions such as `sourceAddress`, `amount`, `feeTokens`, `interval`, and `expiry`. However, the `endTime` parameter is not being checked here, unlike in the `stakeExternal()` method.

Recommendation

Ensure consistency between similar part of the codebase is preserved. You should consider adding a similar validation for `endTime`.

AYN-09 | Redundant Nonce Tracking With Overly Specific Parameters In `claim()` Function

Category	Severity	Location	Status
Logical Issue	● Minor	AyniStaking.sol (base): 415	● Resolved

Description

The contract uses both `salt` and `nonce` for replay protection in the `claim()` function, but the nonce tracking appears redundant and overly restrictive.

```
if (usedSalts[salt]) revert SaltAlreadyUsed();
if (usedNonces[userId][stakeId][nonce]) revert NonceAlreadyUsed();

// ... signature verification ...

usedSalts[salt] = true;
usedNonces[userId][stakeId][nonce] = true;
```

The `salt` alone should be sufficient for replay protection since it's a unique identifier per claim. The additional nonce tracking with `usedNonces[userId][stakeId][nonce]` creates unnecessary complexity and potential issues:

- 1. Redundancy:** Both `salt` and `nonce` serve the same purpose of preventing replay attacks
- 2. Overly restrictive:** If a nonce is intended to be globally unique, why scope it to `userId` and `stakeId`?
- 3. Inconsistency:** The nonce tracking uses `[userId][stakeId][nonce]` while stake storage uses `[userId][interval][stakeId]` - this could lead to confusion and block all the `interval`s for the same user

Recommendation

Simplify the replay protection mechanism by using only `salt`.

AYN-10 | Inconsistent Input Validation Between `addSigner()` And `removeSigner()` Functions

Category	Severity	Location	Status
Inconsistency	● Minor	AyniStaking.sol (base): 456	● Resolved

Description

The `addSigner()` and `removeSigner()` functions have inconsistent input validation patterns:

```
function addSigner(address _signer) external onlyOwner {
    if (_signer == address(0)) revert InvalidInput();
    isSigner[_signer] = true; // No check for existing signer
}

function removeSigner(address _signer) external onlyOwner {
    if (_signer == address(0)) revert InvalidInput();
    if (!isSigner[_signer]) revert InvalidInput(); // Check for existing signer
    isSigner[_signer] = false;
}
```

While `removeSigner()` properly validates that the signer exists before removal, `addSigner()` does not check if the signer already exists.

Recommendation

Add consistent validation to `addSigner()` by checking if the signer already exists.

AYN-11 | Unsafe Fee Collection Pattern

Category	Severity	Location	Status
Logical Issue	● Minor	AyniStaking.sol (base): 328	● Resolved

Description

The `stakeVirtual()` function uses an unsafe fee collection pattern that requires users to approve an unpredictable amount of tokens.

```
328 stakingToken.safeTransferFrom(sourceAddress, feeCollector, feeTokens);
329
330 stakingToken.safeTransferFrom(sourceAddress, address(this), amount);
```

Issues:

- Unpredictable approval requirements:** Users must approve `amount + feeTokens` but cannot know `feeTokens` in advance, requiring them to approve a large arbitrary amount
- Lack of user control:** Users lose control over how much of their approved tokens can be taken as fees

Recommendation

Change the fee collection mechanism to deduct fees from the staked amount rather than requiring separate transfers:

```
uint256 netStakeAmount = amount - feeTokens;
stakes[userId][interval][stakeId] = Stake({
    amount: netStakeAmount,
    // ...
});

stakingToken.safeTransferFrom(sourceAddress, feeCollector, feeTokens);
stakingToken.safeTransferFrom(sourceAddress, address(this), netStakeAmount);
```

This ensures users only need to approve the exact `amount` they intend to stake.

Alleviation

[Ayni, 10/15/2025]: Fee tokens will keep on changing from the backend side and we will let users know how much fee tokens will be deducted through UI. Currently, our backend system is designed to handle it accordingly.

AYN-12 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	● Informational	AyniStaking.sol: 455, 465, 478, 521	● Resolved

Description

There should always be events emitted in sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended to emit events in sensitive functions that are controlled by centralization roles.

AYN-13 | Obsolete And Inaccurate Comments

Category	Severity	Location	Status
Coding Style	● Informational	AyniStaking.sol (base): 162, 263, 364	● Resolved

Description

Some comments are outdated or inaccurate:

- `stakeExternal()` / `stakeVirtual()` comment doesn't mention `expiry` argument
- "feeTokens: amount of tokens to be deducted as gas fee" - in fact, `feeTokens` are not deducted from the `amount`
- `claim()` comment doesn't mention `destinationAddress` argument

Recommendation

We recommend updating the comments.

AYN-14 | Inconsistent Parameter Order Between Encoding And Signature Verification In `claim()` Function

Category	Severity	Location	Status
Inconsistency	● Informational	AyniStaking.sol (base): 407~408	● Resolved

Description

The `claim()` function has inconsistent parameter ordering between the ABI decoding and signature verification processes.

The parameters `salt` and `userId` are swapped between the two operations. This doesn't break functionality as long as the backend consistently uses the same order when generating signatures that the contract uses when verifying them.

Recommendation

For better code clarity and maintainability, standardize the parameter ordering between ABI decoding and signature construction.

Alleviation

[Ayni, 10/15/2025]: The current parameter order is intentional and aligns with the backend's implementation for ABI encoding and signature generation. Although the order differs between the ABI decoding and signature construction, both systems are designed to use the same sequence consistently. To maintain compatibility with the backend, it is necessary to retain the existing order.

AYN-15 | Potentially Misleading Event Parameter Name

`lastPreclaimMonth` Contains Unclear "Pre" Prefix

Category	Severity	Location	Status
Coding Style	● Informational	AyniStaking.sol (base): 445	● Resolved

Description

The event parameter name `lastPreclaimMonth` contains the prefix "pre" which creates confusion about its meaning.

```
emit Claimed(  
    destinationAddress,  
    userId,  
    stakeId,  
    interval,  
    claimedMonth, // Should match the event parameter name  
    rewards,  
    userStake.amount  
);
```

The "pre" prefix suggests this might refer to something that happens **before** the actual claim, but based on the context where this value updates `userStake.claimedUntilMonth`, it actually represents the **last month for which rewards are being claimed** in the current transaction.

The prefix "pre" could be interpreted as:

- A month before the actual claim period
- A preliminary claim month
- A pre-claim state

This naming ambiguity could confuse developers and off-chain systems trying to understand the exact meaning of this parameter.

Recommendation

Consider renaming the event parameter to either:

1. `lastClaimedMonth` - clearly indicates this is the final month being claimed
2. `claimedUntilMonth` - matches the storage variable name for consistency
3. Remove the "pre" prefix if it serves no meaningful purpose in the context

OPTIMIZATIONS | AYNİ

ID	Title	Category	Severity	Status
AYN-02	NotMatured Error Is Never Used	Code Optimization	Optimization	● Resolved

AYN-02 | `NotMatured` Error Is Never Used

Category	Severity	Location	Status
Code Optimization	● Optimization	AyniStaking.sol (base): 106	● Resolved

Description

`NotMatured` error is never used.

Recommendation

We recommend removing of unused code.

APPENDIX | AYNI

Finding Categories

Categories	Description
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your **Web3** Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is the largest blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

