



AXC

Security Assessment

CertiK Assessed on Dec 8th, 2025





Certik Assessed on Dec 8th, 2025

AXC

The security assessment was prepared by Certik.

Executive Summary

TYPES

DeFi

ECOSYSTEM

Binance Smart Chain
(BSC)

METHODS

Formal Verification, Manual Review, Static Analysis

LANGUAGE

Solidity

TIMELINE

Preliminary comments published on 11/21/2025

Final report published on 12/08/2025

Vulnerability Summary



17

Total Findings

13

Resolved

0

Partially Resolved

4

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.

1 Major

1 Resolved



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

1 Medium

1 Acknowledged



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

6 Minor

6 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.

7 Informational

6 Resolved, 1 Acknowledged



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 | AXC

Summary

[Executive Summary](#)

[Vulnerability Summary](#)

[Codebase](#)

[Audit Scope](#)

[Approach & Methods](#)

Review Notes

[Overview](#)

[Core Contracts](#)

[Crowdsale](#)

[ShareToken](#)

[FundVault](#)

[FundYield](#)

[External Dependencies](#)

[Privileged Functions](#)

Findings

[AXC-02 : Centralized Control Of Contract Upgrade](#)

[AXC-03 : Centralization Related Risks](#)

[AXC-07 : Potential Signature Replay Due To Incomplete Signature Data](#)

[AXC-17 : Concern On Off-Chain Operations](#)

[AXC-04 : Lack Of Storage Gap Or NameSpaced Storage Layout In Upgradeable Contract](#)

[AXC-09 : Division By Zero In Calculation](#)

[AXC-10 : Missing Zero Address Validation](#)

[AXC-19 : Potential Template Count Inconsistency](#)

[AXC-20 : Incompatibility With Deflationary Tokens](#)

[AXC-23 : Potential Precision Loss](#)

[AXC-11 : Risks Of Mixing `Ownable` And `AccessControl` In Smart Contract Permissions](#)

[AXC-12 : `setRoleAdmin` To `DEFAULT\\_ADMIN\\_ROLE` Is Redundant](#)

[AXC-13 : Inherited Contracts Not Initialized In Initializer](#)

[AXC-14 : Inaccurate Variable Naming](#)

[AXC-16 : Incorrect Amount Emitted In `RedemptionClaim` Event](#)

AXC-22 : Redundant Import Files

AXC-24 : Start Time Not Enforced To Be In The Future

I Formal Verification

Considered Functions And Scope

Verification Results

I Appendix

I Disclaimer

CODEBASE | AXC

Repository

<https://github.com/accelerate-protocol/accelerate-contract/tree/7006eea361f50520a617442d1d67783ec01be237/contracts/v2>

Commit

Base: [7006eea361f50520a617442d1d67783ec01be237](#)

Final: [1297333894fa1b698bc3dd72db808b5a3c2ada31](#)

Audit Scope

The file in scope is listed in the appendix.

APPROACH & METHODS | AXC

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

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.

REVIEW NOTES | AXC

Overview

The **AXC** protocol provides fundraising, share issuance, and epoch-based yield distribution via upgradeable contracts, with role-based access control to securely manage deposits, redemptions, and fund operations.

Core Contracts

Crowdsale

The contract implements a flexible fundraising mechanism with on-chain and off-chain deposits. It supports share token minting through a vault and enforces funding periods, soft/hard caps, and management fees. Access control is handled via multiple roles for managers, pausers, and off-chain operators. Deposits and redemptions require validated signatures to prevent unauthorized actions and replay attacks. The contract also provides functions for withdrawing raised funds and management fees, pausing/resuming operations, and querying funding status, total raised assets, and user nonces.

ShareToken

The contract is the vault's share certificate token, built on ERC20 with pausable functionality. Only the Vault contract is authorized to mint, burn, and control the pause state. The token uses an initialization pattern to set metadata and decimals and remains paused during the funding period. All transfers trigger a Vault callback via `onTokenTransfer()` once initialized. The design ensures controlled issuance, restricted privileged operations, and seamless integration with the Vault's accounting logic.

FundVault

The contract acts as the system's core storage and permission layer, handling roles, whitelist enforcement, pausing, and references to the Token, Funding, and Yield modules. It enables minting, burning, and transfer hooks through authorized modules. During initialization, it configures managers, validators, whitelist settings, and module addresses. The contract provides robust access control, pause functionality, and reentrancy protection to secure all interactions.

FundYield

The contract manages redemption and yield distribution for a `FundVault`. Users can stake share tokens per epoch, request or cancel redemptions, and claim rewards once epochs are liquidated. Managers control epoch transitions, set parameters, and settle rewards with validated signatures.

External Dependencies

The **AXC** protocol relies on a few external dependencies to fulfill the needs of its business logic.

The following are third-party dependency contracts used within the contract:

- @openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol
- @openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol

- @openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol
- @openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol
- @openzeppelin/contracts-upgradeable/utils/PausableUpgradeable.sol
- @openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol
- @openzeppelin/contracts/utils/cryptography/ECDSA.sol
- @openzeppelin/contracts/utils/cryptography/MessageHashUtils.sol
- @openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol
- @openzeppelin/contracts/token/ERC20/IERC20.sol
- @openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol
- @openzeppelin/contracts/access/Ownable.sol
- @openzeppelin/contracts/proxy/transparent/ProxyAdmin.sol
- @openzeppelin/contracts/proxy/transparent/TransparentUpgradeableProxy.sol
- @openzeppelin/contracts/access/AccessControl.sol

It is assumed that these functions are trusted and implemented properly within the whole project.

Additionally, it is important to note that the audit scope excludes tokens not created by this platform; these are treated as third-party dependencies. Similarly, external contracts such as the vault, yield, and crowdsale functionalities, which were created by third parties, are also outside the scope of this review.

Privileged Functions

In the **AXC** protocol, some privileged roles are adopted to ensure the dynamic runtime updates of the project, which are specified in the **Centralization Related Risks** and **Centralized Control of Contract Upgrade** findings.

The advantage of those privileged roles in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community.

It is also worth noting the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the private keys of the privileged accounts are compromised, it could lead to devastating consequences for the project.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should also be considered to move to the execution queue of the `TimeLock` contract.

FINDINGS | AXC



17
Total Findings

0
Critical

2
Centralization

1
Major

1
Medium

6
Minor

7
Informational

This report has been prepared for AXC to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the audit, a total of 17 issues were identified. Leveraging a combination of Static Analysis & Manual Review the following findings were uncovered:

ID	Title	Category	Severity	Status
AXC-02	Centralized Control Of Contract Upgrade	Centralization	Centralization	● Acknowledged
AXC-03	Centralization Related Risks	Centralization	Centralization	● Acknowledged
AXC-07	Potential Signature Replay Due To Incomplete Signature Data	Logical Issue	Major	● Resolved
AXC-17	Concern On Off-Chain Operations	Design Issue	Medium	● Acknowledged
AXC-04	Lack Of Storage Gap Or NameSpaced Storage Layout In Upgradeable Contract	Design Issue	Minor	● Resolved
AXC-09	Division By Zero In Calculation	Logical Issue	Minor	● Resolved
AXC-10	Missing Zero Address Validation	Volatile Code	Minor	● Resolved
AXC-19	Potential Template Count Inconsistency	Logical Issue	Minor	● Resolved
AXC-20	Incompatibility With Deflationary Tokens	Volatile Code	Minor	● Resolved
AXC-23	Potential Precision Loss	Logical Issue	Minor	● Resolved

ID	Title	Category	Severity	Status
AXC-11	Risks Of Mixing <code>Ownable</code> And <code>AccessControl</code> In Smart Contract Permissions	Design Issue	Informational	● Resolved
AXC-12	<code>_setRoleAdmin</code> To <code>DEFAULT_ADMIN_ROLE</code> IS Redundant	Logical Issue	Informational	● Resolved
AXC-13	Inherited Contracts Not Initialized In Initializer	Logical Issue	Informational	● Resolved
AXC-14	Inaccurate Variable Naming	Coding Style	Informational	● Resolved
AXC-16	Incorrect Amount Emitted In <code>RedemptionClaim</code> Event	Coding Issue	Informational	● Resolved
AXC-22	Redundant Import Files	Coding Style	Informational	● Resolved
AXC-24	Start Time Not Enforced To Be In The Future	Logical Issue	Informational	● Acknowledged

AXC-02 | Centralized Control Of Contract Upgrade

Category	Severity	Location	Status
Centralization	● Centralization	templates/funding/Crowdsale.sol (7006eea361f50520a617442d1d67783ec01be237): 32; templates/token/ShareToken.sol (7006eea361f50520a617442d1d67783ec01be237): 24; templates/vault/FundVault.sol (7006eea361f50520a617442d1d67783ec01be237): 27; templates/yield/FundYield.sol (7006eea361f50520a617442d1d67783ec01be237): 29	● Acknowledged

Description

In the contracts `Crowdsale`, `ShareToken`, `FundVault` and `FundYield`, the role `admin` has the authority to update the implementation contract behind these contracts.

Any compromise to the `admin` account may allow a hacker to take advantage of this authority and change the implementation contract, which is pointed to by proxy, and therefore execute potentially malicious functionality in the implementation contract.

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (2/3, 3/5) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

Short Term:

A combination of a time-lock and a multi signature (2/3, 3/5) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
AND
- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

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

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

I Alleviation

[AXC, 12/02/2025]:

This risk originates from the management of privileged private keys used in operating the RWA protocol. We take this potential risk very seriously and have implemented strict measures to ensure the security and decentralized management of these privileged keys:

1. The protocol's proxy administrator rights are not held by a single externally owned account (EOA). Instead, they are delegated to a multi-signature wallet, ensuring that no single party can execute privileged actions unilaterally.

2. A timelock contract is introduced as the guardian address for upgrade operations, adding an additional layer of protection and transparency.

To maintain community trust and transparency, we will publicly disclose all key privileged addresses at the time of project launch, enabling community members to independently verify and monitor their activity. Furthermore, to advance the decentralization of the project, we plan to gradually introduce DAO governance and voting mechanisms in future phases to address this issue in a more trustless and community-driven manner.

[CertiK, 12/03/2025]:

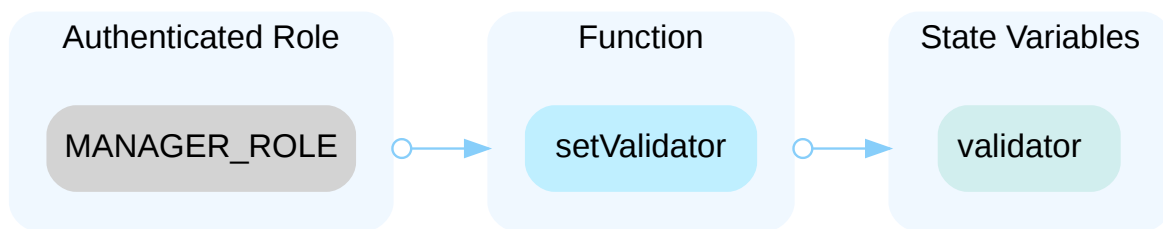
It is suggested to implement the aforementioned methods to avoid centralized failure. CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

AXC-03 | Centralization Related Risks

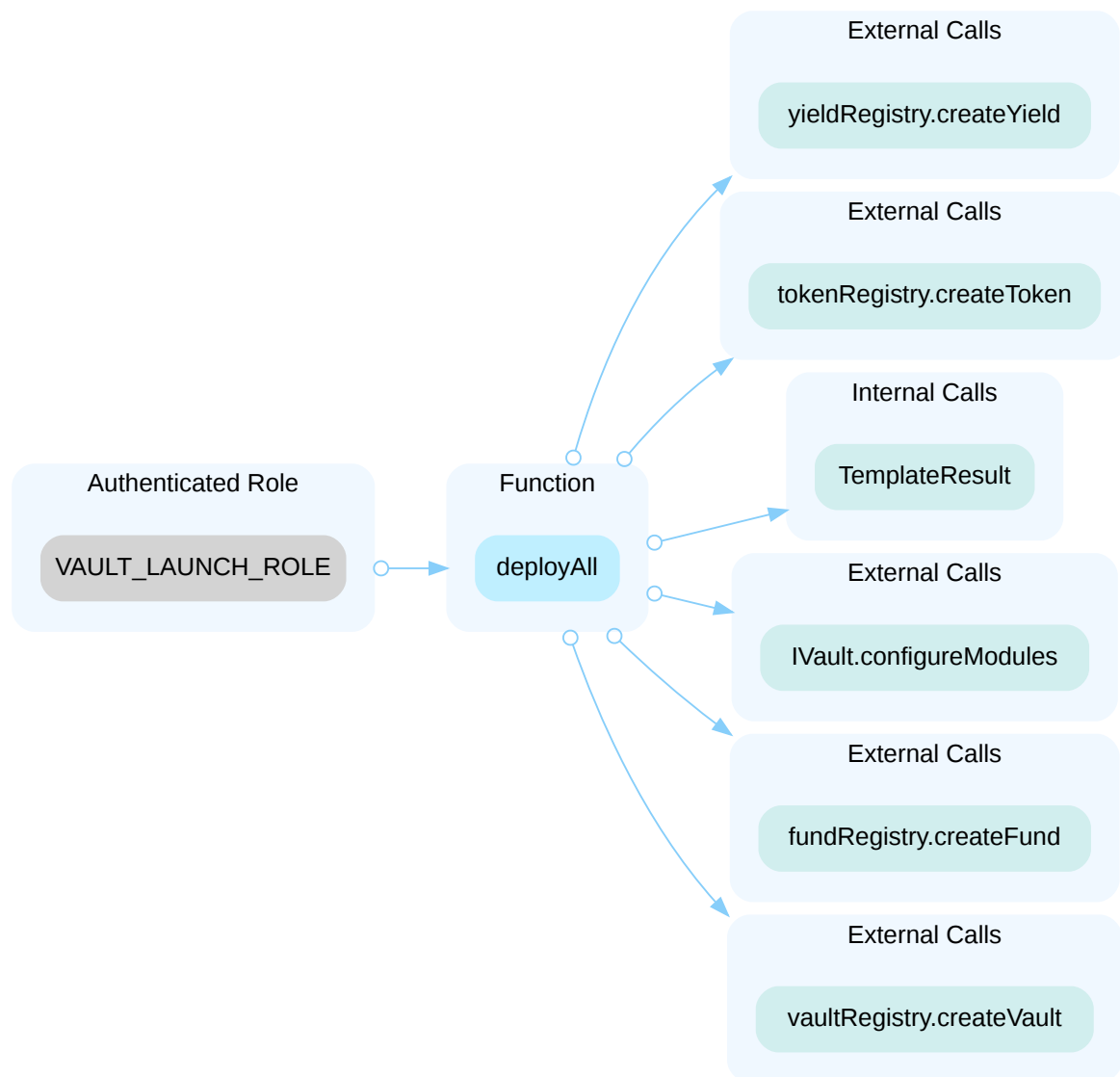
Category	Severity	Location	Status
Centralization	● Centralization		● Acknowledged

Description

In the contract `ValidatorRegistry`, the role `MANAGER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `MANAGER_ROLE` account may allow the hacker to take advantage of this authority and change the validator address.



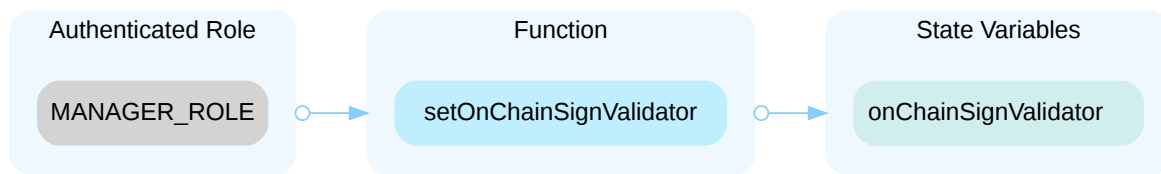
In the contract `Creation`, the role `VAULT_LAUNCH_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `VAULT_LAUNCH_ROLE` account may allow the hacker to take advantage of this authority and deploy `Crowdsale`, `ShareToken`, `FundVault` and `FundYield` contracts.



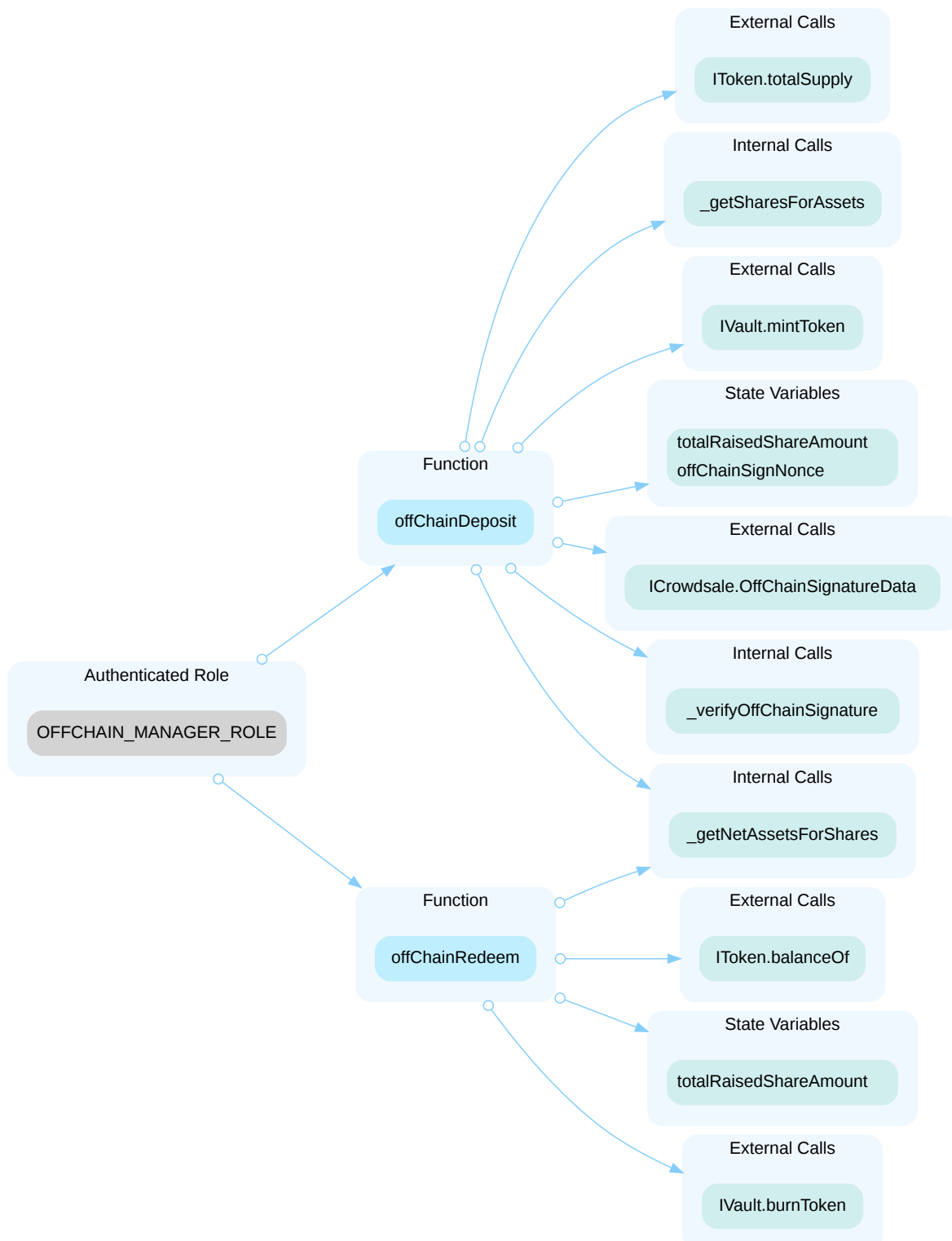
In the contracts `FundTemplateRegistry`, `TokenTemplateRegistry`, `VaultTemplateRegistry` and `YieldTemplateRegistry`, the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and add contract templates.



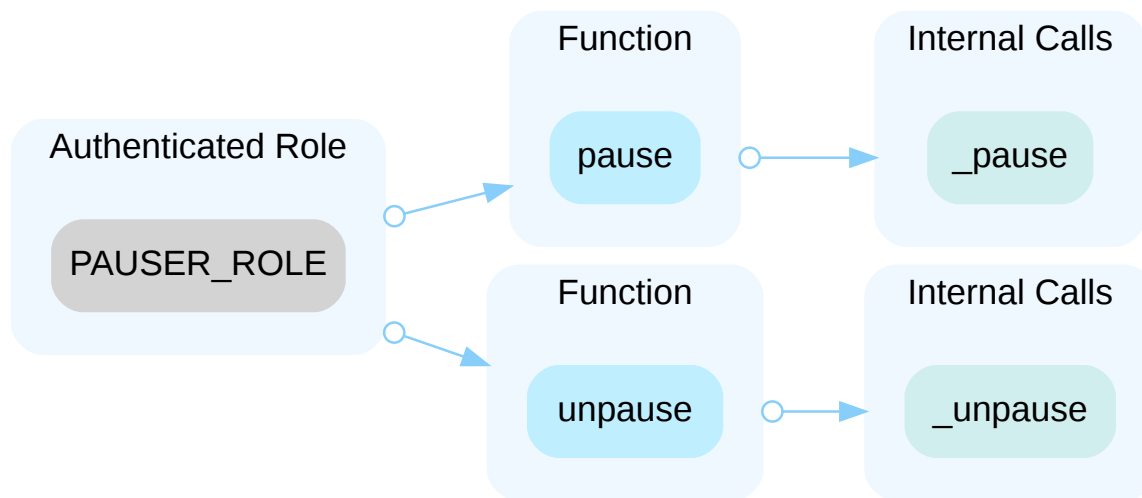
In the contract `Crowdsale`, the role `MANAGER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `MANAGER_ROLE` account may allow the hacker to take advantage of this authority and set an on-chain signature validator address.



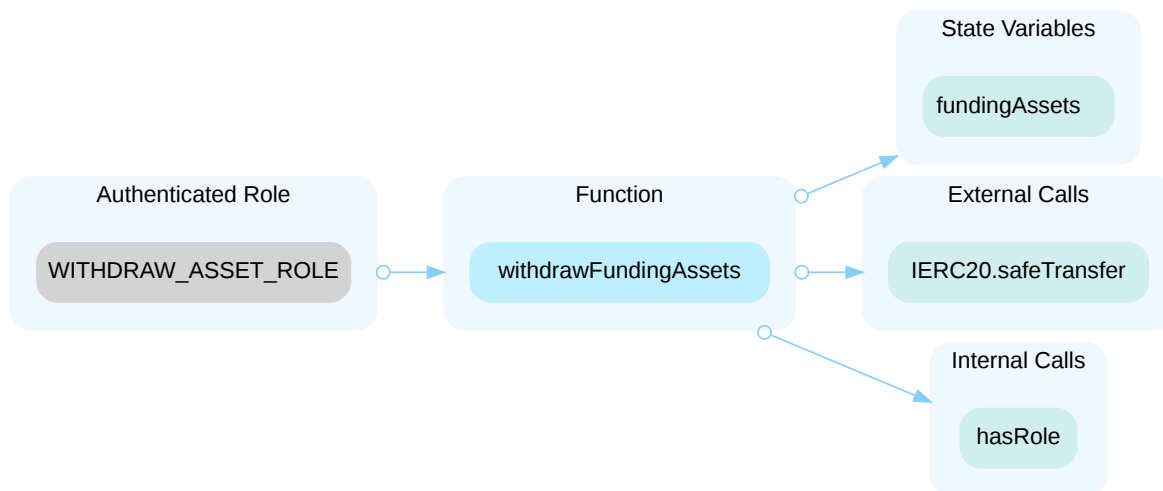
In the contract `Crowdsale`, the role `OFFCHAIN_MANAGER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `OFFCHAIN_MANAGER_ROLE` account may allow the hacker to take advantage of this authority and create deposit data for the specific address and redeem the deposit.



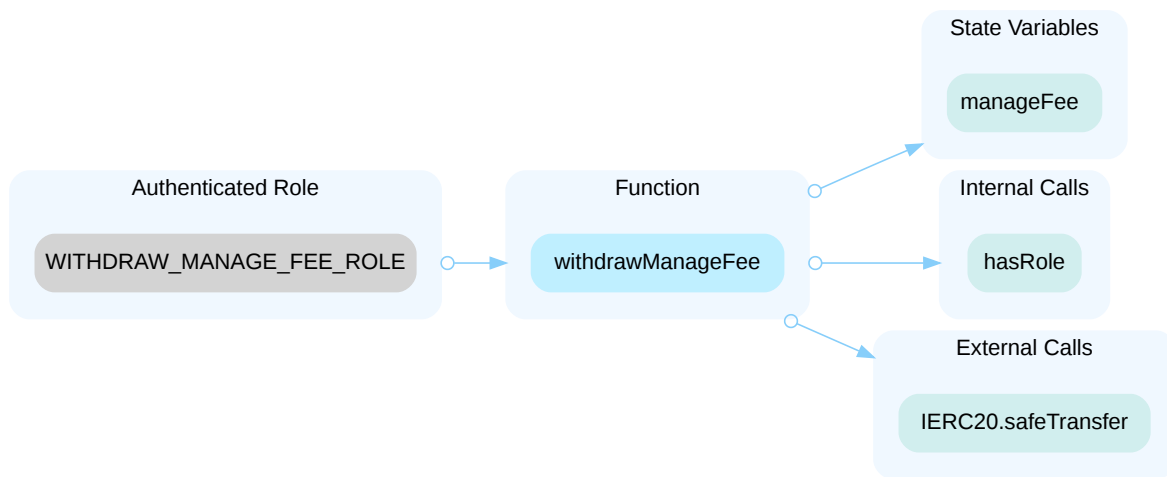
In the contract `Crowdsale`, the role `PAUSER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `PAUSER_ROLE` account may allow the hacker to take advantage of this authority and pause/unpause the contract.



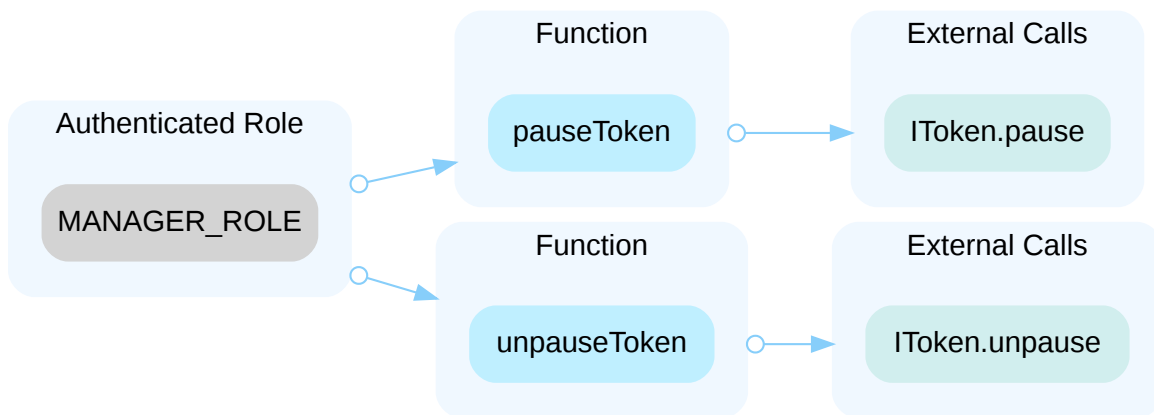
In the contract `Crowdsale`, the roles `WITHDRAW_ASSET_ROLE` and `fundingReceiver` have authority over the functions shown in the diagram below. Any compromise to the `WITHDRAW_ASSET_ROLE` and `fundingReceiver` accounts may allow the hacker to take advantage of this authority and withdraw funding assets.



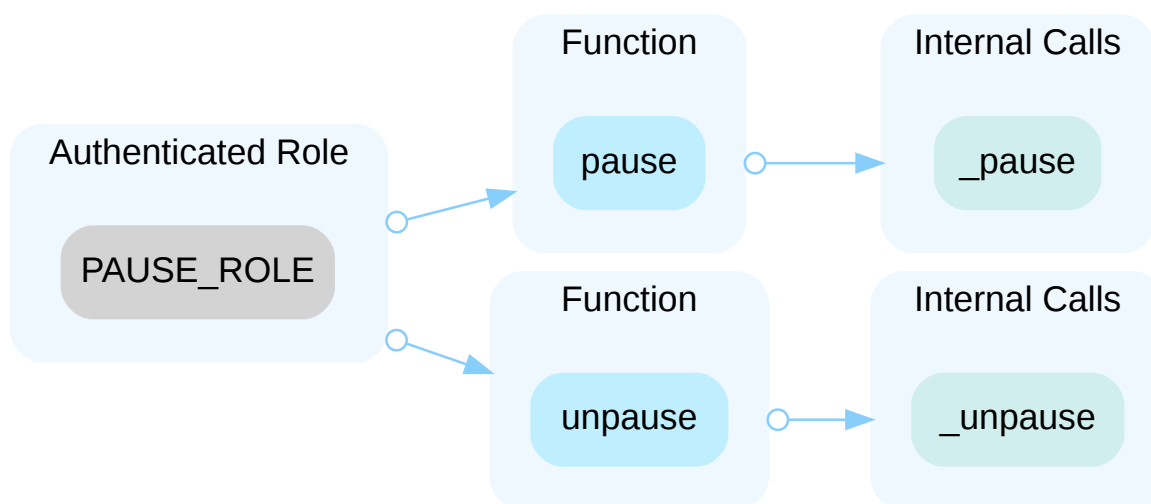
In the contract `Crowdsale`, the roles `WITHDRAW_MANAGE_FEE_ROLE` and `manageFeeReceiver` have authority over the functions shown in the diagram below. Any compromise to the `WITHDRAW_MANAGE_FEE_ROLE` and `manageFeeReceiver` accounts may allow the hacker to take advantage of this authority and withdraw the management fee.



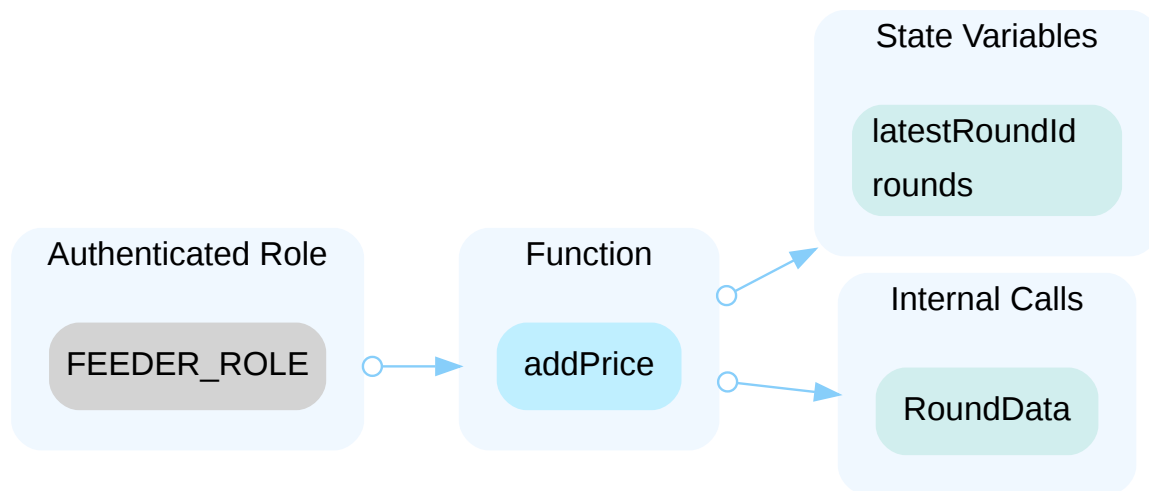
In the contract `BaseVault`, the role `MANAGER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `MANAGER_ROLE` account may allow the hacker to take advantage of this authority and pause/unpause the token contract.



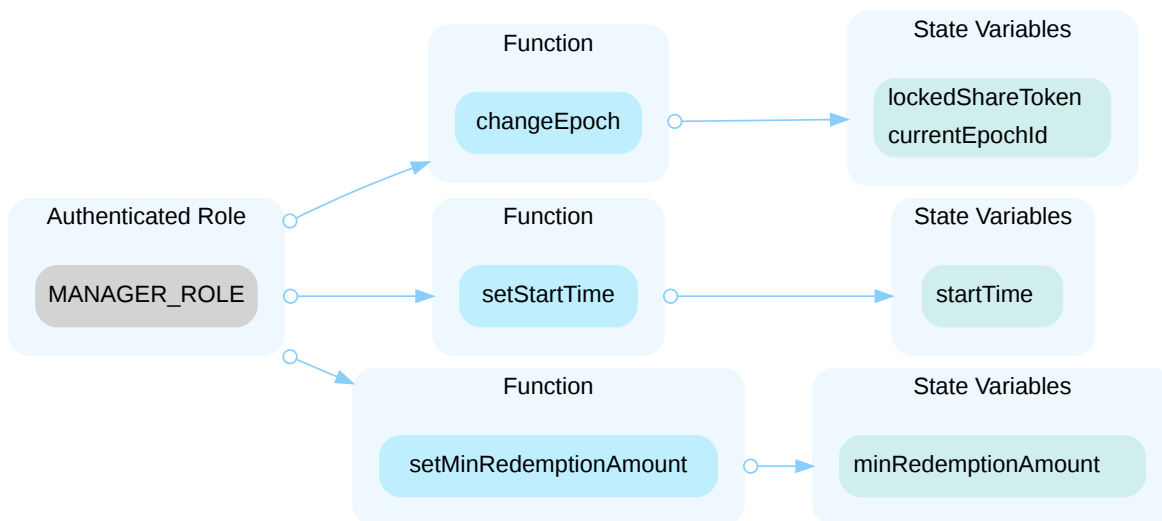
In the contract `BaseVault`, the role `PAUSE_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `PAUSE_ROLE` account may allow the hacker to take advantage of this authority and pause/unpause the contract.



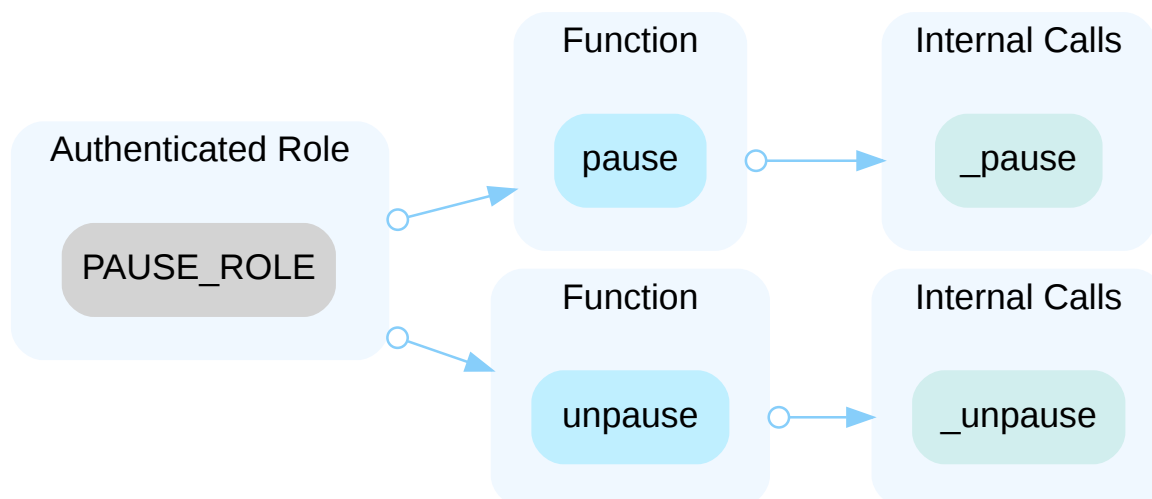
In the contract `FundVault`, the role `FEEDER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `FEEDER_ROLE` account may allow the hacker to take advantage of this authority and add share token price data.



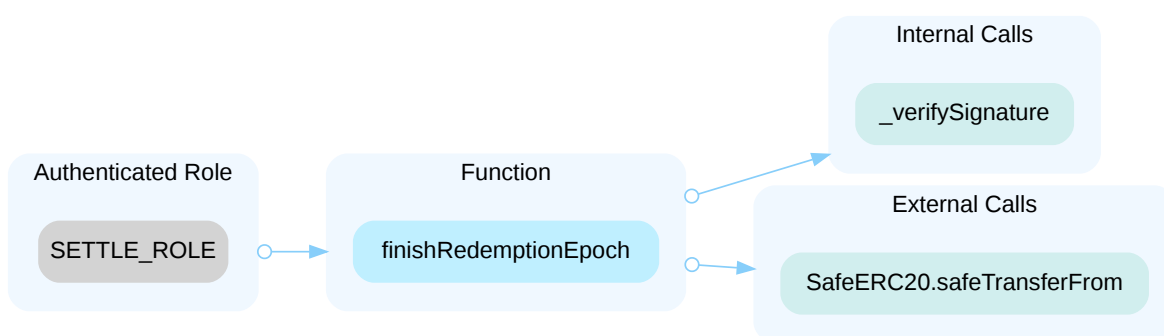
In the contract `FundYield`, the role `MANAGER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `MANAGER_ROLE` account may allow the hacker to take advantage of this authority and change to the next epoch, update the start time, and set the `minRedemptionAmount`.



In the contract `FundYield`, the role `PAUSE_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `PAUSE_ROLE` account may allow the hacker to take advantage of this authority and pause/unpause the contract.



In the contract `FundYield`, the role `SETTLE_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `SETTLE_ROLE` account may allow the hacker to take advantage of this authority, finish the epoch and transfer rewards to the contract.



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 (2/3, 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

[AXC, 12/02/2025]:

This risk originates from the management of privileged private keys used in operating the RWA protocol. We take this potential risk very seriously and have implemented strict measures to ensure the security and decentralized management of these privileged keys:

1. The protocol's proxy administrator rights are not held by a single externally owned account (EOA). Instead, they are delegated to a multi-signature wallet, ensuring that no single party can execute privileged actions unilaterally.
2. A timelock contract is introduced as the guardian address for upgrade operations, adding an additional layer of protection and transparency.

To maintain community trust and transparency, we will publicly disclose all key privileged addresses at the time of project launch, enabling community members to independently verify and monitor their activity. Furthermore, to advance the decentralization of the project, we plan to gradually introduce DAO governance and voting mechanisms in future phases to address this issue in a more trustless and community-driven manner.

[CertiK, 12/03/2025]:

It is suggested to implement the aforementioned methods to avoid centralized failure. CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

AXC-07 | Potential Signature Replay Due To Incomplete Signature Data

Category	Severity	Location	Status
Logical Issue	● Major	templates/funding/Crowdsale.sol (7006eea361f50520a617442d1d67783ec01be237): 154, 235; templates/yield/FundYield.sol (7006eea361f50520a617442d1d67783ec01be237): 222	● Resolved

Description

The `_verifySignature()` function in the `Crowdsale` contract currently constructs the `OnChainSignatureData` structure without incorporating the caller's address (`msg.sender`) into the signed message.

Although the `nonce` is maintained per caller, the signature only binds to the included fields (`operation`, `amount`, `receiver`, `nonce`, `chainId`, and `contractAddress`). If two separate users produce transactions with matching parameter values and identical per-user nonces, a valid signature generated for one user can be replayed by another. This could enable unauthorized users to perform token redemptions or deposits using a signature that was intended for a different participant, leading to potential misuse of token operations. Including an identifier unique to the caller in the signed message would ensure that signatures cannot be validly replayed for users other than the original intended signer.

```
function deposit(uint256 amount, address receiver, bytes memory signature)
    external
    override
    onlyInitialized
    onlyDuringFunding
    nonReentrant
    whenNotPaused
{
    ...
    uint256 nonce = callerNonce[msg.sender]++;

    ICrowdsale.OnChainSignatureData memory sigData =
    ICrowdsale.OnChainSignatureData({
        operation: "deposit",
        amount: amount,
        receiver: receiver,
        nonce: nonce,
        chainId: block.chainid,
        contractAddress: address(this)
    });

    _verifySignature(sigData, signature);
}
```

Similarly, both the `redeem()` function and the `finishRedemptionEpoch()` function in the `FundYield` contract use relatively minimal signed payloads. Although these functions rely on other mechanisms that effectively prevent signature replay, it is still recommended to strengthen the signed message structure to ensure that signatures are inherently non-reusable across all contexts.

I Proof of Concept

This test case simulates two accounts with the same nonce successfully invoking the `deposit` function using an identical signature. If the protocol includes any quota or allocation constraints, a malicious user could leverage multiple accounts that share the same nonce to obtain more shares than intended, bypassing the designed limits.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console, Vm} from "forge-std/Test.sol";
import "../src/v2/common/ValidatorRegistry.sol";
import {FundTemplateRegistry} from
"../src/v2/templateRegistry/FundTemplateRegistry.sol";
import {TokenTemplateRegistry} from
"../src/v2/templateRegistry/TokenTemplateRegistry.sol";
import {VaultTemplateRegistry} from
"../src/v2/templateRegistry/VaultTemplateRegistry.sol";
import {YieldTemplateRegistry} from
"../src/v2/templateRegistry/YieldTemplateRegistry.sol";
import {FundYield} from "../src/v2/templates/yield/FundYield.sol";
import {FundVault} from "../src/v2/templates/vault/FundVault.sol";
import {ShareToken} from "../src/v2/templates/token/ShareToken.sol";
import {Crowdsale, ICrowdsale} from "../src/v2/templates/funding/Crowdsale.sol";
import {MessageHashUtils} from
"../@openzeppelin/contracts/utils/cryptography/MessageHashUtils.sol";
import {Creation, ICreation} from "../src/v2/creation/Creation.sol";
import {ERC20} from "../@openzeppelin/contracts/token/ERC20/ERC20.sol";
import {CrowdsaleTemplateFactory} from
"../src/v2/factories/funding/CrowdsaleTemplateFactory.sol";
import {ShareTokenTemplateFactory} from
"../src/v2/factories/token/ShareTokenTemplateFactory.sol";
import {FundVaultTemplateFactory} from
"../src/v2/factories/vault/FundVaultTemplateFactory.sol";
import {FundYieldTemplateFactory} from
"../src/v2/factories/yield/FundYieldTemplateFactory.sol";

contract MockToken is ERC20 {
    constructor(string memory name_, string memory symbol_)ERC20(name_, symbol_) {
    }
}

contract CounterTest is Test {
    ValidatorRegistry validatorRegistry;
    FundTemplateRegistry fundTemplateRegistry;
    TokenTemplateRegistry tokenTemplateRegistry;
    VaultTemplateRegistry vaultTemplateRegistry;
    YieldTemplateRegistry yieldTemplateRegistry;
    Creation creation;
    MockToken rewardToken = new MockToken("RWT", "RWT");
    MockToken assetToken = new MockToken("USDC", "USDC");

    address validator = address(0x1);
    uint256 managerKey = 0xA11CE;
    address manager = vm.addr(managerKey);
    address guardian = address(0x3);
```

```
address settleRole = address(0x4);
address fundingReceiver = address(0x5);
address manageFeeReceiver = address(0x6);
address offchainManager = address(0x7);

address alice = address(0x123);
address bob = address(0x234);
address cindy = address(0x345);

address public vault;
address public token;
address public yield;
address public fund;

function setUp() public {
    validatorRegistry = new ValidatorRegistry(validator, manager);
    fundTemplateRegistry = new FundTemplateRegistry();
    tokenTemplateRegistry = new TokenTemplateRegistry();
    vaultTemplateRegistry = new VaultTemplateRegistry();
    yieldTemplateRegistry = new YieldTemplateRegistry();
    address[] memory managers = new address[](1);
    managers[0] = manager;
    creation = new Creation(address(vaultTemplateRegistry),
address(tokenTemplateRegistry), address(fundTemplateRegistry),
address(yieldTemplateRegistry), managers);

    add_template();
    deploy_all();
}

function deploy_all() public {

    address[] memory initialWhitelist = new address[](3);
    initialWhitelist[0] = alice;
    initialWhitelist[1] = bob;
    initialWhitelist[2] = cindy;
    bytes memory _vaultInitData = abi.encode(manager,
address(validatorRegistry), true, initialWhitelist);

    string memory name = "AXC";
    string memory symbol = "AXC";
    uint8 decimals = 18;
    bytes memory _tokenInitData = abi.encode(name, symbol, decimals);

    uint256 minRedemptionAmount = 10e18;
    uint256 startTime = block.timestamp;
    bytes memory _yieldInitData = abi.encode(address(rewardToken), manager,
settleRole, minRedemptionAmount, startTime);
```

```
uint256 endTime = startTime + 30 days;           // Crowdsale end time
uint256 maxSupply = 10000e18;                     // Maximum number of shares to be
sold in share token units
uint256 softCap = 6000e18;                         // Minimum amount to be raised for
successful funding
uint256 sharePrice = 2e8;                         // Price per share in asset token units,
scaled by 1e8
uint256 minDepositAmount = 100e18;               // Minimum deposit amount per
transaction in asset token units
uint256 manageFeeBps = 500;                      // Management fee in basis points (bps)

bytes memory _fundInitData = abi.encode(
    startTime,
    endTime,
    address(assetToken),
    maxSupply,
    softCap,
    sharePrice,
    minDepositAmount,
    manageFeeBps,
    fundingReceiver,
    manageFeeReceiver,
    manager,
    offchainManager
);

ICreation.DeployParams memory params = ICreation.DeployParams({
    vaultTemplateId: 1,
    vaultInitData: _vaultInitData,
    tokenTemplateId: 1,
    tokenInitData: _tokenInitData,
    fundTemplateId: 1,
    fundInitData: _fundInitData,
    yieldTemplateId: 1,
    yieldInitData: _yieldInitData,
    guardian: guardian
});
creation.deployAll(params);
vault = creation.vaultC();
token = creation.tokenC();
yield = creation.yieldC();
fund = creation.fundC();
}

function test_deposit() public {
    vm.startPrank(alice);
    deal(address(assetToken), alice, 100e18);
    assetToken.approve(fund, 100e18);
```

```
ICrowdsale.OnChainSignatureData memory sigData =
ICrowdsale.OnChainSignatureData({
    operation: "deposit",
    amount: 100e18,
    receiver: alice,
    nonce: 0,
    chainId: block.chainid,
    contractAddress: fund
});

bytes32 ethSignedMessageHash = data_hash(sigData);
(uint8 v, bytes32 r, bytes32 s) = vm.sign(managerKey, ethSignedMessageHash);
bytes memory signature = abi.encodePacked(r, s, v);

Crowdsale(fund).deposit(100e18, alice, signature);

vm.startPrank(bob);
deal(address(assetToken), bob, 100e18);
assetToken.approve(fund, 100e18);
Crowdsale(fund).deposit(100e18, alice, signature);

vm.warp(block.timestamp + 30 days + 1);

vm.startPrank(alice);
sigData = ICrowdsale.OnChainSignatureData({
    operation: "redeem",
    amount: ShareToken(token).balanceOf(alice),
    receiver: alice,
    nonce: 1,
    chainId: block.chainid,
    contractAddress: fund
});

ethSignedMessageHash = data_hash(sigData);
(v, r, s) = vm.sign(managerKey, ethSignedMessageHash);
signature = abi.encodePacked(r, s, v);
console.log("alice share:", ShareToken(token).balanceOf(alice)/1e8);
ShareToken(token).approve(vault, ShareToken(token).balanceOf(alice));
Crowdsale(fund).redeem(ShareToken(token).balanceOf(alice), alice,
signature);
console.log("alice balance:", assetToken.balanceOf(alice)/1e18);

}

function data_hash(ICrowdsale.OnChainSignatureData memory sigData) public pure
returns(bytes32){
    bytes32 messageHash = keccak256(abi.encodePacked(
        sigData.operation,
        sigData.amount,
```

```
        sigData.receiver,  
        sigData.nonce,  
        sigData.chainId,  
        sigData.contractAddress  
    ));  
    bytes32 ethSignedMessageHash =  
    MessageHashUtils.toEthSignedMessageHash(messageHash);  
    return ethSignedMessageHash;  
}  
  
function add_template() public {  
    CrowdsaleTemplateFactory crowdsale = new CrowdsaleTemplateFactory();  
    ShareTokenTemplateFactory shareToken = new ShareTokenTemplateFactory();  
    FundVaultTemplateFactory fundVault = new FundVaultTemplateFactory();  
    FundYieldTemplateFactory fundYield = new FundYieldTemplateFactory();  
  
    fundTemplateRegistry.addTemplate(1, address(crowdsale));  
    tokenTemplateRegistry.addTemplate(1, address(shareToken));  
    vaultTemplateRegistry.addTemplate(1, address(fundVault));  
    yieldTemplateRegistry.addTemplate(1, address(fundYield));  
}  
}
```

Output is:

```
Ran 1 test for test/Test.t.sol:CounterTest  
[PASS] test_deposit() (gas: 1064776)  
Logs:  
  alice share: 950000000000  
  alice balance: 200  
  
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 5.96ms (2.63ms CPU  
time)
```

Recommendation

It is recommended to review the signature generation and verification logic to ensure that signatures cannot be reused across different users or contexts. Consider including relevant identifiers in the signed data to strengthen uniqueness.

Alleviation

[AXC, 12/03/2025]:

The team heeded the advice and resolved the issue by completing the chain, contract, identity, and other relevant information in the signed message in commit [57786feec4b820a8bb929ac1760568c2e2d9794e](#).

AXC-17 | Concern On Off-Chain Operations

Category	Severity	Location	Status
Design Issue	● Medium	templates/funding/Crowdsale.sol (7006eea361f50520a617442d1d67783ec01be237): 273	● Acknowledged

Description

The `Crowdsale` contract offers two distinct methods for users to interact with share tokens: a standard on-chain deposit-redeem flow and an off-chain process managed by accounts with the `OFFCHAIN_MANAGER_ROLE`.

In the on-chain flow, users deposit `assetToken` to receive shares and can redeem shares to withdraw the equivalent amount of `assetToken`. With the off-chain mechanism, an authorized manager can mint shares to any address through `offChainDeposit()` without requiring an actual deposit of `assetToken`, and can also burn shares via `offChainRedeem()`.

A potential concern arises if a user obtains shares that were minted off-chain (without backing asset deposits) and later acquires a valid on-chain authorization or signature that allows them to redeem these shares for `assetToken`. This would enable the user to withdraw tokens from the contract that were never actually deposited, reducing the contract's `assetToken` reserve. As a result, users who participated through the standard deposit process could be unable to redeem their shares for `assetToken`, given the diminished pool.

Recommendation

We would like the team to clarify the intended use cases for the off-chain mechanism and whether the scenario described above could occur in practice.

Alleviation

[AXC, 12/02/2025]:

In the current protocol, the subscription process includes both on-chain and off-chain participation. The off-chain service listens to and records how each subscriber acquires RWA tokens during the fundraising phase. If the fundraising fails, users must obtain an attestation signature from the off-chain service validator before initiating an on-chain redemption. The validator verifies whether the user's RWA tokens originated from on-chain subscription.

For RWA assets subscribed off-chain with fiat currency, redemption requires authorization and assistance from the off-chain administrator, who burns the corresponding tokens and facilitates the fiat refund. Therefore, an off-chain subscriber cannot redeem through the on-chain redemption mechanism in a way that prevents legitimate on-chain subscribers from redeeming their assets.

[CertiK, 12/08/2025]:

Since obtaining shares off-chain does not involve the transfer of `assetToken` into the contract, we recommend that clients

sign carefully to prevent users from using off-chain acquired shares to directly redeem `assetToken` from the contract.

AXC-04 Lack Of Storage Gap Or NameSpaced Storage Layout In Upgradeable Contract

Category	Severity	Location	Status
Design Issue	Minor	templates/vault/BaseVault.sol (7006eea361f50520a617442d1d67783ec01be237): 27	Resolved

Description

When updating upgradeable smart contracts for new features or bug fixes, keeping the state variables' declaration order unchanged is essential to avoid storage layout issues.

A practical solution is to include unused state variables or explicitly named storage gaps (like `__gap`) in the base contracts. This foresight allows reserved slots for future use, ensuring that any additions to the contract's state won't disrupt the storage pattern of derived contracts or the compatibility with previously deployed versions. After [ERC-7201](#), it is also possible to place all storage variables of a contract into one or more structs like `Namespaced Storage Layout`.

The problem of "Lack of Storage Gap Or NameSpaced Storage Layout in Upgradeable Contract" occurs when **these storage gaps are not incorporated into the base contract's logic nor the base contract defines the namespace storage layout**. As a result, if new state variables are added to the base contract, they might overwrite existing variables in the child contracts due to storage slot collisions.

In the current contract, the contract allows for future upgrades and is also inherited by other contracts. However, the storage gap is missing for the the current contract, nor is the `namespaced storage layout` used.

For detailed guidelines and best practices, refer to the following OpenZeppelin documentation:

- https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gaps
- <https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#storage-gaps>

Recommendation

To mitigate this issue:

- For enhanced flexibility in future upgrades of the logic contract, it is prudent to reserve a storage gap of an appropriate size in the base contract. This is achieved by declaring a fixed-size array, typically of `uint256` elements, each occupying a 32-byte slot, in the base contract. Label this array with the identifier `__gap` or any name prefixed with `__gap` to indicate its purpose as a reserved space clearly.
- it is also possible by placing all storage variables of a contract into one or more structs like `Namespaced Storage Layout`.

More detailed info :

- https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gaps

I Alleviation

[AXC, 12/03/2025]:

The team heeded the advice and resolved the issue by adding storage gaps in commit
[57786feec4b820a8bb929ac1760568c2e2d9794e](#).

AXC-09 | Division By Zero In Calculation

Category	Severity	Location	Status
Logical Issue	Minor	templates/funding/Crowdsale.sol (7006eea361f50520a617442d1d67783ec01be237): 180, 466, 573	Resolved

Description

According to the design, `manageFeeBps` is required to satisfy when initializing:

```
require(data.manageFeeBps <= BPS_DENOMINATOR, "Crowdsale: invalid manage fee")
```

However, if `manageFeeBps == BPS_DENOMINATOR` (i.e., 100%), the reverse fee calculation will encounter:

```
function _getManagerFeeWithShares(uint256 shareAmount) internal view returns
(uint256) {
    uint256 netAssetAmount = _getNetAssetsForShares(shareAmount);
    @> uint256 managerFee = (netAssetAmount * manageFeeBps) / (BPS_DENOMINATOR -
manageFeeBps);
    return managerFee;
}

function deposit(uint256 amount, address receiver, bytes memory signature)
    external override onlyInitialized onlyDuringFunding nonReentrant
whenNotPaused
{
    ...
    if (requestedShares > remainingSupply) {
        actualShares = remainingSupply;
        uint256 actualNetAmount = _getNetAssetsForShares(remainingSupply);
    @> actualAmount = (actualNetAmount * BPS_DENOMINATOR) / (BPS_DENOMINATOR -
manageFeeBps);
        require(actualAmount >= minDepositAmount, "Crowdsale: remaining amount
below minimum");
    } else {
        actualShares = requestedShares;
        actualAmount = amount;
    }
    ...
}
```

This leads to a division-by-zero error, causing any functionality relying on this calculation to fail.

Recommendation

We recommend enforcing `manageFeeBps < BPS_DENOMINATOR` to prevent the management fee from being set to 100%, ensuring the reverse calculation cannot revert due to division by zero.

■ Alleviation

[AXC, 12/02/2025]:

The team heeded the advice and resolved the issue by removing the `=` condition in the requirement in commit [ade82a4b6747b32901aae71a3ee1d150b374623d](#).

AXC-10 | Missing Zero Address Validation

Category	Severity	Location	Status
Volatile Code	Minor	common/ValidatorRegistry.sol (7006eea361f50520a617442d1d67783ec01be237): 29; templates/funding/Crowdsale.sol (7006eea361f50520a617442d1d67783ec01be237): 501; templates/vault/BaseVault.sol (7006eea361f50520a617442d1d67783ec01be237): 250	Resolved

Description

The cited address input is missing a check that it is not `address(0)`. If a zero address is provided as an input parameter, it may result in the corresponding role being missing.

ValidatorRegistry

```

    constructor(address _validator, address _manager) Ownable(msg.sender) {
        require(_validator != address(0), "ValidatorRegistry: invalid validator address");
        validator = _validator;
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _grantRole(MANAGER_ROLE, msg.sender);
    @>    _grantRole(MANAGER_ROLE, _manager);
        _setRoleAdmin(MANAGER_ROLE, DEFAULT_ADMIN_ROLE);
    }

```

Crowdsale

```

    function _initCrowdsale(
        CrowdsaleInitializedData memory data
    ) internal {
        ...
    @>    _grantRole(OFFCHAIN_MANAGER_ROLE, data.offchainManager);
    }

```

BaseVault

```

    function _setYieldModule(address _yieldModule) internal virtual {
        require(yield == address(0), "Vault: dividend module already set");
    @>    yield = _yieldModule;
    }

```

Recommendation

We recommend adding a check the passed-in address is not `address(0)` to prevent unexpected errors.

I Alleviation

[AXC, 12/02/2025]:

The team heeded the advice and resolved the issue by adding the zero address validation in commit [ade82a4b6747b32901aae71a3ee1d150b374623d](#).

AXC-19 | Potential Template Count Inconsistency

Category	Severity	Location	Status
Logical Issue	Minor	templateRegistry/FundTemplateRegistry.sol (7006eea361f50520a617442d1d67783ec01be237): 29; templateRegistry/TokenTemplateRegistry.sol (7006eea361f50520a617442d1d67783ec01be237): 29; templateRegistry/VaultTemplateRegistry.sol (7006eea361f50520a617442d1d67783ec01be237): 29; templateRegistry/YieldTemplateRegistry.sol (7006eea361f50520a617442d1d67783ec01be237): 30	Resolved

Description

The `addTemplate` function updates `templateCount` based on the condition `templateId >= templateCount`. This logic assumes that `templateId` values are added in sequential order. If a non-sequential `templateId` (e.g., jumping from 3 to 100) is added, `templateCount` becomes `templateId + 1`, resulting in an incorrect `templateCount`.

```
function addTemplate(uint256 templateId, address template) external override
onlyOwner {
    require(templateId != 0, "FundTemplateRegistry: invalid templateId");
    require(template != address(0), "FundTemplateRegistry: invalid template");
    require(templates[templateId] == address(0), "FundTemplateRegistry: template
already exists");

    templates[templateId] = template;
    if (templateId >= templateCount) {
@>        templateCount = templateId + 1;
    }
    emit TemplateAdded(templateId, template);
}
```

Recommendation

We recommend incrementing the template count by 1 each time a new template is added to ensure the template count remains accurate.

Alleviation

[AXC, 12/03/2025]:

The team heeded the advice and resolved the issue by incrementing the template count by 1 each time a new template is added in commit [57786feec4b820a8bb929ac1760568c2e2d9794e](#).

AXC-20 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Volatile Code	● Minor	templates/funding/Crowdsale.sol (7006eea361f50520a617442d1d67783ec01be237): 196~200; templates/yield/FundYield.sol (7006eea361f50520a617442d1d67783ec01be237): 230~235	● Resolved

Description

When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. As a result, an inconsistency in the amount will occur, and the transaction may fail due to the validation checks. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee) to the target contract, only 90 tokens actually arrive at the contract.

Crowdsale

```
function deposit(uint256 amount, address receiver, bytes memory signature)
    external override onlyInitialized onlyDuringFunding nonReentrant
whenNotPaused
{
    ...
    // Transfer assets (user pays the full actual amount, fee is deducted)
    @> IERC20(assetToken).safeTransferFrom(
        msg.sender,
        address(this),
        actualAmount
    );
    ...
}
```

FundYield

```
function finishRedemptionEpoch(
    uint256 epochId,
    uint256 assetAmount,
    bytes memory signature
) external whenNotPaused active onlyInitialized nonReentrant
onlyRole(SETTLE_ROLE)
{
    ...
    @> SafeERC20.safeTransferFrom(
        IERC20(rewardToken),
        msg.sender,
        address(this),
        assetAmount
    );
    emit RedemptionEpochLiquidated(
        msg.sender,
        epochId,
        assetAmount,
        signature
    );
}
```

Recommendation

We recommend validating the actual amount of tokens received rather than relying solely on the requested transfer value. Aligning internal accounting with the true post-transfer balance can help prevent inconsistencies caused by fee-on-transfer or deflationary tokens.

Alleviation

[AXC, 12/02/2025]:

The team heeded the advice and resolved the issue by validating the actual amount of tokens received in commit [ade82a4b6747b32901aae71a3ee1d150b374623d](#).

AXC-23 | Potential Precision Loss

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/v2/templates/funding/Crowdsale.sol (12/02/2025-ade82a4): 264, 375, 390	● Resolved

Description

Users interact with the `Crowdsale` contract by depositing `assertToken` to purchase share tokens. In the event of a failed funding activity, users are able to redeem their `assertToken`. Conversely, if the funding succeeds, privileged roles can withdraw the investment or associated fees. The calculation of either the redeemable amount or the withdrawable amount relies on formulas that may introduce precision loss due to integer truncation.

This precision loss may cause the following issues:

- The calculated amount may differ from the records (`fundingAssets` , `manageFee`), and the last users to redeem may be unable to withdraw all `assetToken` due to the underflow error.
- The calculated amount might not align with the contract's actual token balance, which could lead to a failure in redeeming or withdrawing the investment or associated fees.
- The last users cannot get back all their investment.

```
function redeem(uint256 amount, address receiver, bytes memory signature)
    external
    override
    onlyInitialized
    onlyAfterFundingFailed
    nonReentrant
    whenNotPaused
{
    ...

    // Update state
    @> fundingAssets -= netAssetAmount;
    @> manageFee -= feeAmount;
    // Refund assets (including management fee)
    @> IERC20(assetToken).safeTransfer(receiver, grossAssetAmount);

    emit FundFailedRedeem(msg.sender, receiver, amount, netAssetAmount,
    feeAmount);
}

function withdrawFundingAssets() external override onlyInitialized
onlyAfterFundingSuccess nonReentrant whenNotPaused {
    require(msg.sender == fundingReceiver || hasRole(WITHDRAW_ASSET_ROLE,
    msg.sender), "Crowdsale: unauthorized");
    require(fundingAssets > 0, "Crowdsale: no funding assets");

    uint256 amount = fundingAssets;
    fundingAssets = 0;

    @> IERC20(assetToken).safeTransfer(fundingReceiver, amount);

    emit FundingAssetsWithdrawn(msg.sender, fundingReceiver, amount);
}

function withdrawManageFee() external override onlyInitialized
onlyAfterFundingSuccess nonReentrant whenNotPaused {
    require(msg.sender == manageFeeReceiver || hasRole(WITHDRAW_MANAGE_FEE_ROLE,
    msg.sender), "Crowdsale: unauthorized");
    require(manageFee > 0, "Crowdsale: no manage fee");

    uint256 amount = manageFee;
    manageFee = 0;

    @> IERC20(assetToken).safeTransfer(manageFeeReceiver, amount);
}
```

```
emit ManageFeeWithdrawn(msg.sender, manageFeeReceiver, amount);  
}
```

Recommendation

It's recommended that redemption and withdrawal logic be updated to ensure that the actual amount of `assertToken` held by the contract is transferred, especially in cases where calculated amounts are affected by precision loss and do not match the contract's token balance or records. The logic should safeguard against transaction failures and the overflow error.

For example, if the calculated amount exceeds the contract's balance, the transfer uses the contract's actual balance. If it exceeds the recorded amount, the record is set to zero. This approach will prevent failures and align actual transfers with the contract's token holdings.

Alleviation

[AXC, 12/08/2025]:

The team heeded the advice and resolved the issue by adding the necessary checks to prevent the process from failing in commit [1297333894fa1b698bc3dd72db808b5a3c2ada31](#).

[CertiK, 12/08/2025]:

It is advisable to notify the community that due to negligible precision variances in the calculation, there is a remote possibility that the contract's balance could be marginally insufficient, causing the very last users to redeem to receive less than their initial investment.

AXC-11 | Risks Of Mixing Ownable And AccessControl In Smart Contract Permissions

Category	Severity	Location	Status
Design Issue	● Informational	common/ValidatorRegistry.sol (7006eea361f50520a617442d1d67783ec01be237): 15; creation/Creation.sol (7006eea361f50520a617442d1d67783ec01be237): 22; templates/funding/Crowdsale.sol (7006eea361f50520a617442d1d67783ec01be237): 32; templates/token/ShareToken.sol (7006eea361f50520a617442d1d67783ec01be237): 12; templates/yield/FundYield.sol (7006eea361f50520a617442d1d67783ec01be237): 29	● Resolved

Description

The contract's inheritance of both `Ownable` and `AccessControl` introduces additional complexity in permission management because each base contract provides a different mechanism for managing permissions and access rights.

- `Ownable` is a simpler model where there is a single account (the owner) with the authority to perform administrative actions.
- `AccessControl`, on the other hand, provides a more flexible and granular permission system based on roles, which can be assigned to different accounts.

The overlap of a singular ownership model with a multi-role access system can complicate the understanding and governance of the contract, increasing the risk of misconfiguration and security vulnerabilities.

Recommendation

To mitigate this issue, a clear rationale for integrating both mechanisms should be provided, alongside comprehensive testing to ensure their coexistence does not compromise the contract's security and intended functionality.

Alleviation

[AXC, 12/08/2025]:

The team heeded the advice and resolved the issue by removing the inheritance of the `Ownable` contract in commit [1297333894fa1b698bc3dd72db808b5a3c2ada31](#).

AXC-12 | `_setRoleAdmin` To `DEFAULT_ADMIN_ROLE` Is Redundant

Category	Severity	Location	Status
Logical Issue	 Informational	common/ValidatorRegistry.sol (7006eea361f50520a617442d1d67783ec01be237): 30; creation/Creation.sol (7006eea361f50520a617442d1d67783ec01be237): 75; templates/vault/BaseVault.sol (7006eea361f50520a617442d1d67783ec01be237): 293	 Resolved

Description

The specific role is assigned with `DEFAULT_ADMIN_ROLE` as the role admin. However, the assignment here is redundant, as `DEFAULT_ADMIN_ROLE` will be the role admin for any role by default.

Recommendation

It is recommended to remove the redundant assignment.

Alleviation

[AXC, 12/03/2025]:

The team heeded the advice and resolved the issue by removing the redundant assignment in commit [57786feec4b820a8bb929ac1760568c2e2d9794e](#).

AXC-13 | Inherited Contracts Not Initialized In Initializer

Category	Severity	Location	Status
Logical Issue	● Informational	templates/funding/Crowdsale.sol (7006eea361f50520a617442d1d67783ec01be237): 32, 456	● Resolved

Description

Contract `Crowdsale` extends `AccessControlUpgradeable`, but the extended contract is not initialized by the current contract. Generally, the initializer function of a contract should always call all the initializer functions of the contracts that it extends.

Recommendation

We recommend initializing `AccessControlUpgradeable`.

Alleviation

[AXC, 12/02/2025]:

The team heeded the advice and partially resolved the issue by adding the initialization of the `AccessControlUpgradeable` contract in commit [ade82a4b6747b32901aae71a3ee1d150b374623d](#).

AXC-14 | Inaccurate Variable Naming

Category	Severity	Location	Status
Coding Style	● Informational	factories/yield/FundYieldTemplateFactory.sol (7006eea361f50520a617442d1d67783ec01be237): 31	● Resolved

Description

The local variable is named `vaultProxy`, but it actually refers to a `yieldProxy`, which can cause confusion for developers reading or maintaining the code. Using a misleading variable name may reduce code clarity and increase the chance of misunderstandings or errors during future updates.

Recommendation

It's recommended that the variable currently named `vaultProxy` be renamed to `yieldProxy` to accurately reflect its purpose and type, improving code clarity and reducing the risk of confusion or errors in future development and maintenance.

Alleviation

[AXC, 12/02/2025]:

The team heeded the advice and resolved the issue by renaming the param name in commit [ade82a4b6747b32901aae71a3ee1d150b374623d](#).

AXC-16 | Incorrect Amount Emitted In `RedemptionClaim` Event

Category	Severity	Location	Status
Coding Issue	● Informational	interfaces/templates/IFundYield.sol (7006eea361f50520a617442d1d67783ec01be237): 47; templates/yield/FundYield.sol (7006eea361f50520a617442d1d67783ec01be237): 276~281	● Resolved

Description

The last two values supplied to the `RedemptionClaim` event are passed in reverse order compared to the parameter names defined in the event signature. This causes the amounts reported in the event logs to be inaccurate, resulting in potential confusion for off-chain consumers or analytics that rely on correct event data to monitor redemptions.

```
event RedemptionClaim(address requester,uint256 epochId,uint256
shareAmount,uint256 assetAmount);

function claimRedemption(
    uint256 epochId
) external whenNotPaused active onlyInitialized nonReentrant {
    ...
    emit RedemptionClaim(
        msg.sender,
        epochId,
        @> claimAssets,
        @> request.requestShares
    );
}
```

Recommendation

We recommend modifying the event parameters to ensure that the events are correctly emitted.

Alleviation

[AXC, 12/02/2025]:

The team heeded the advice and resolved the issue by modifying the event parameters in commit [ade82a4b6747b32901aae71a3ee1d150b374623d](#).

AXC-22 | Redundant Import Files

Category	Severity	Location	Status
Coding Style	● Informational	templates/funding/Crowdsale.sol (7006eea361f50520a617442d1d67783ec01be237): 23~24; templates/vault/BaseVault.sol (7006eea361f50520a617442d1d67783ec01be237): 19~20; templates/yield/FundYield.sol (7006eea361f50520a617442d1d67783ec01be237): 21~22	● Resolved

Description

Multiple import statements in the contracts reference the same files more than once, leading to redundant imports. This can reduce code clarity and maintainability. Removing these duplicate imports will help keep the codebase clean and consistent.

Recommendation

We recommend removing the duplicate imports for code consistency.

Alleviation

[AXC, 12/02/2025]:

The team heeded the advice and resolved the issue by removing the duplicate imports in commit [ade82a4b6747b32901aae71a3ee1d150b374623d](#).

AXC-24 | Start Time Not Enforced To Be In The Future

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/v2/templates/funding/Crowdsale.sol (12/02/2025-a de82a4): 464	● Acknowledged

Description

In the `Crowdsale` contract, the `_initCrowdsale()` function only checks `startTime < endTime` and `endTime > block.timestamp`, but does not require `startTime > block.timestamp`. As a result, the crowdsale may be initialized in an already-active state, skipping the intended full participation window or allowing premature participation.

```
function _initCrowdsale(
    CrowdsaleInitializedData memory data
) internal {
    require(data.startTime < data.endTime, "Crowdsale: invalid time range");
    @> require(data.endTime > block.timestamp, "Crowdsale: end time in past");
    ...
}
```

Recommendation

Consider adding a validation to ensure the crowdsale enters a complete and controlled activity period after initialization.

Alleviation

[AXC, 12/08/2025]:

The design of the funding module does not enforce that the start time must be strictly later than the current block timestamp. This allows the off-chain system to deploy a project that starts immediately or has technically already begun. In addition, to avoid deployment rejections caused by potential time discrepancies between off-chain requests and on-chain block timestamps, the start time is intentionally left unrestricted for now.

FORMAL VERIFICATION | AXC

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of contracts derived from AccessControl v4.4

We verified properties of the public interface of contracts that provide an AccessControl-v4.4 compatible API. This involves:

- The `hasRole` function, which returns `true` if an account has been granted a specific `role`.
- The `getRoleAdmin` function, which returns the admin role that controls a specific `role`.
- The `grantRole` and `revokeRole` functions, which are used for granting a `role` to an account and revoking a `role` from an `account`, respectively.
- The `renounceRole` function, which allows the calling account to revoke a `role` from itself.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
accesscontrol-getroleadmin-succeed-always	<code>getRoleAdmin</code> Function Always Succeeds
accesscontrol-hasrole-succeed-always	<code>hasRole</code> Function Always Succeeds
accesscontrol-hasrole-change-state	<code>hasRole</code> Function Does Not Change State
accesscontrol-getroleadmin-change-state	<code>getRoleAdmin</code> Function Does Not Change State
accesscontrol-renouncerole-revert-not-sender	<code>renounceRole</code> Reverts When Caller Is Not the Confirmation Address
accesscontrol-renouncerole-succeed-role-renouncing	<code>renounceRole</code> Successfully Renounces Role
accesscontrol-grantrole-correct-role-granting	<code>grantRole</code> Correctly Grants Role
accesscontrol-default-admin-role	AccessControl Default Admin Role Invariance
accesscontrol-revokerole-correct-role-revoking	<code>revokeRole</code> Correctly Revokes Role

Verification of Standard Ownable Properties

We verified *partial* properties of the public interfaces of those token contracts that implement the Ownable interface. This involves:

- function `owner` that returns the current owner,
- functions `renounceOwnership` that removes ownership,
- function `transferOwnership` that transfers the ownership to a new owner.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
ownable-transferownership-correct	Ownership is Transferred
ownable-renounceownership-correct	Ownership is Removed
ownable-owner-succeed-normal	<code>owner</code> Always Succeeds
ownable-renounce-ownership-is-permanent	Once Renounced, Ownership Cannot be Regained

Verification Results

For the following contracts, formal verification established that each of the properties that were in scope of this audit (see scope) are valid:

Detailed Results For Contract ValidatorRegistry (contracts/v2/common/ValidatorRegistry.sol) In Commit 7006eea361f50520a617442d1d67783ec01be237

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	● True	
accesscontrol-hasrole-change-state	● True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	● True	
accesscontrol-renouncerole-succeed-role-renouncing	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results For Contract Creation (contracts/v2/creation/Creation.sol) In Commit 7006eea361f50520a617442d1d67783ec01be237**Verification of contracts derived from AccessControl v4.4**Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-succeed-role-renouncing	● True	
accesscontrol-renouncerole-revert-not-sender	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	● True	
accesscontrol-hasrole-change-state	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results For Contract FundTemplateRegistry
(contracts/v2/templateRegistry/FundTemplateRegistry.sol) In Commit
7006eea361f50520a617442d1d67783ec01be237

Verification of Standard Ownable Properties

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	● True	

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	● True	
ownable-renounce-ownership-is-permanent	● True	

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	● True	

Detailed Results For Contract TokenTemplateRegistry (contracts/v2/templateRegistry/TokenTemplateRegistry.sol) In Commit 7006eea361f50520a617442d1d67783ec01be237

Verification of Standard Ownable Properties

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	● True	

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	● True	

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounce-ownership-is-permanent	● True	
ownable-renounceownership-correct	● True	

**Detailed Results For Contract VaultTemplateRegistry
(contracts/v2/templateRegistry/VaultTemplateRegistry.sol) In Commit
7006eea361f50520a617442d1d67783ec01be237**

Verification of Standard Ownable Properties

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	● True	
ownable-renounce-ownership-is-permanent	● True	

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	● True	

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	● True	

**Detailed Results For Contract YieldTemplateRegistry
(contracts/v2/templateRegistry/YieldTemplateRegistry.sol) In Commit
7006eea361f50520a617442d1d67783ec01be237**

Verification of Standard Ownable Properties

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	● True	

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	● True	

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	● True	
ownable-renounce-ownership-is-permanent	● True	

APPENDIX | AXC

Audit Scope

accelerate-protocol/accelerate-contract

	templates/funding/Crowdsale.sol
	templates/token/ShareToken.sol
	templates/vault/FundVault.sol
	templates/yield/FundYield.sol
	common/ValidatorRegistry.sol
	creation/Creation.sol
	factories/yield/FundYieldTemplateFactory.sol
	templateRegistry/FundTemplateRegistry.sol
	templateRegistry/TokenTemplateRegistry.sol
	templateRegistry/VaultTemplateRegistry.sol
	templateRegistry/YieldTemplateRegistry.sol
	templates/vault/BaseVault.sol
	factories/funding/CrowdsaleTemplateFactory.sol
	factories/token/ShareTokenTemplateFactory.sol
	factories/vault/FundVaultTemplateFactory.sol

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.

Categories	Description
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
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.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.

- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed AccessControl-v4.4 Properties

Properties related to function `getRoleAdmin`

`accesscontrol-getroleadmin-change-state`

The `getRoleAdmin` function must not change any state variables.

Specification:

```
assignable \nothing;
```

`accesscontrol-getroleadmin-succeed-always`

The `getRoleAdmin` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `hasRole`

`accesscontrol-hasrole-change-state`

The `hasRole` function must not change any state variables.

Specification:

```
assignable \nothing;
```

`accesscontrol-hasrole-succeed-always`

The `hasRole` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `renounceRole`

accesscontrol-renouncerole-revert-not-sender

The `renounceRole` function must revert if the caller is not the same as `account`.

Specification:

```
reverts_when account != msg.sender;
```

accesscontrol-renouncerole-succeed-role-renouncing

After execution, `renounceRole` must ensure the caller no longer has the renounced role.

Specification:

```
ensures !hasRole(role, account);
```

Properties related to function `grantRole`**accesscontrol-grantrole-correct-role-granting**

After execution, `grantRole` must ensure the specified account has the granted role.

Specification:

```
ensures hasRole(role, account);
```

Properties related to function `DEFAULT_ADMIN_ROLE`**accesscontrol-default-admin-role**

The default admin role must be invariant, ensuring consistent access control management.

Specification:

```
invariant DEFAULT_ADMIN_ROLE() == 0x00;
```

Properties related to function `revokeRole`**accesscontrol-revokerole-correct-role-revoking**

After execution, `revokeRole` must ensure the specified account no longer has the revoked role.

Specification:

```
ensures !hasRole(role, account);
```

Description of the Analyzed Ownable Properties

Properties related to function `transferOwnership`

ownable-transferownership-correct

Invocations of `transferOwnership(newOwner)` must transfer the ownership to the `newOwner`.

Specification:

```
ensures this.owner() == newOwner;
```

Properties related to function `renounceOwnership`

ownable-renounce-ownership-is-permanent

The contract must prohibit regaining of ownership once it has been renounced.

Specification:

```
constraint \old(owner()) == address(0) ==> owner() == address(0);
```

ownable-renounceownership-correct

Invocations of `renounceOwnership()` must set ownership to `address(0)`.

Specification:

```
ensures this.owner() == address(0);
```

Properties related to function `owner`

ownable-owner-succeed-normal

Function `owner` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

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.

