Cyfrin Attester
Security Review

MARCH, 2025

www.chaindefenders.xyz
https://x.com/DefendersAudits

Lead Auditors

PeterSR

0x539.eth

Table of Contents

Protocol Summary

A set of smart contracts intended to enable Cyfrin to make attestations about anything onchain. Examples include: Certifications, KYC, Badges, Achievements, etc. It uses the Ethereum Attestation Service (EAS) to make these attestations, specifically from the `CyfrinAttester` contract.

The first attestation that we plan to make is to Cyfrin Solidity Certifications. We've implemented an EAS resolver, `CyfrinSolidityCert`, which acts as a resolver to this

Certification schema. Upon attesting, the resolver mints a Soulbound, expirable NFT.

Disclaimer

The Chain Defenders team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

Likelihood/Impact	High	Medium	Low
High	H	H/M	M
Medium	H/M	M	M/L
Low	M	M/L	L

Audit Details

Scope

Id	Files in scope
1	CyfrinAttester.sol
2	Withdrawable.sol
3	CyfrinSolidityCert.sol
4	Soulbound.sol
5	IAdmin.sol
6	ICyfrinAttester.sol
7	ICyfrinSolidityCert.sol
8	IERC5192.sol
9	IERC5484.sol
10	IERC7572.sol

Roles

Id	Roles
1	Owner
2	DEFAULT_ADMIN_ROLE
3	ATTESTER_ROLE
4	EAS
5	User

Executive Summary

Issues found

Severity	Count	Description
High	1	Critical vulnerabilities
Medium	2	Significant risks

Severity	Count	Description
Low	6	Minor issues with low impact
Informational	4	Best practices or suggestions
Gas	0	Optimization opportunities

Findings

High

High 01 Incompatibility With Some Chains

Summary

The `CyfrinAttester` contract is incompatible with some of the target chains (Arbitrum One, Base, Optimism and Ethereum). This is due to the fact that the version of the `EAS` contract deployed on these chains has different implementations of the `DelegatedAttestationRequest` and `DelegatedRevocationRequest` structs.

Vulnerability Details

The `CyfrinAttester` contract is incompatible with some of the target chains (Arbitrum One, Base, Optimism and Ethereum). This is due to the fact that the version of the `EAS` contract deployed on these chains has different implementations of the `DelegatedAttestationRequest` and `DelegatedRevocationRequest` structs.

Imported structs:

```
struct DelegatedAttestationRequest {
    bytes32 schema; // The unique identifier of the schema.
    AttestationRequestData data; // The arguments of the attestation
   request.
    Signature signature; // The ECDSA signature data.
    address attester; // The attesting account.
    uint64 deadline; // The deadline of the signature/request.
}

struct DelegatedRevocationRequest {
    bytes32 schema; // The unique identifier of the schema.
```

```solidity
    RevocationRequestData data; // The arguments of the revocation
    request.
    Signature signature; // The ECDSA signature data.
    address revoker; // The revoking account.
    uint64 deadline; // The deadline of the signature/request.
}
```

Arbitrum One, Base, Optimism and Ethereum version's structs:

```solidity
struct DelegatedAttestationRequest {
    bytes32 schema; // The unique identifier of the schema.
    AttestationRequestData data; // The arguments of the attestation
    request.
    Signature signature; // The ECDSA signature data.
    address attester; // The attesting account.
}

struct DelegatedRevocationRequest {
    bytes32 schema; // The unique identifier of the schema.
    RevocationRequestData data; // The arguments of the revocation
    request.
    Signature signature; // The ECDSA signature data.
    address revoker; // The revoking account.
}
```

Both the `DelegatedAttestationRequest` and `DelegatedRevocationRequest` structs lack a `deadline` parameter. This makes the current implementation of `CyfrinAttester` incompatible with some of the target chains as all calls to `revokeByDelegation` and `attestByDelegation` will revert due to the added `deadline` parameter in the request.

You can find the deployed `EAS` contract versions here

Impact

The inability of `CyfrinAttester` to fully support all target chains hinders its functionality on these chains leading to unexpected reverts and DoS of the delegation functionalities. This will make some of the deployed versions of `CyfrinAttester` contract not fully usable.

Tools Used

Manual Review

Recommendations

Either rethink the supported/target chains or generate another version of the `CyfrinAttester` which will import the right `DelegatedAttestationRequest` and `DelegatedRevocationRequest` structs (without the `deadline` parameter) and will support the listed chains.

Medium

Mid 01 Non Working Delegation Logic

Summary

The delegation logic in the `CyfrinAttester` contract is not functioning as intended. The attestation and revocation by delegation require a signature, but in the current setup, the `CyfrinAttester` contract cannot generate the necessary signature.

Vulnerability Details

The `CyfrinAttester` contract includes functions for attestation and revocation by delegation, which require a valid signature to authorize the actions. However, since the CyfrinAttester contract itself is supposed to act as the attester for the `CyfrinSolidityCert` contract, it cannot generate the required signature. This makes the delegation functionality non-operational in its current form.

```
struct DelegatedAttestationRequest {
    bytes32 schema; // The unique identifier of the schema.
    AttestationRequestData data; // The arguments of the attestation
   request.
    Signature signature; // The ECDSA signature data.
    address attester; // The attesting account.
    uint64 deadline; // The deadline of the signature/request.
}

struct DelegatedRevocationRequest {
    bytes32 schema; // The unique identifier of the schema.
    RevocationRequestData data; // The arguments of the revocation
   request.
    Signature signature; // The ECDSA signature data.
    address revoker; // The revoking account.
    uint64 deadline; // The deadline of the signature/request.
}
```

Impact

The inability to perform attestation and revocation by delegation limits the flexibility and usability of the contract. Delegation is a key feature that allows third parties to perform actions on behalf of the attester, and without it, the contract's functionality is significantly reduced. This could hinder the adoption and effectiveness of the attestation system.

Tools Used

Manual Review

Recommendations

Consider removing the delegation logic if it is not required for your use case. If delegation is necessary, implement a different mechanism that allows the `CyfrinAttester` contract to support it.

Mid 02 Protocol Is Not Compatible With EIP-5484

Summary

The `Soulbound` contract implements `EIP-5484`, which defines an interface for special NFTs with immutable ownership and predetermined immutable burn authorization. According to the EIP, an `Issued` event is expected to be emitted when an NFT is issued:

> On issuing, an `Issued` event will be emitted alongside EIP-721's `Transfer` event. This design maintains backward compatibility while providing clear signals to third parties that this is a Soulbound token issuance event.

However, the contract does not emit the `Issued` event when a token is issued.

Also based on the specification of `EIP-5484`

> The issuer SHALL NOT change metadata after issuance.

But in the current implementation, the `baseUri` can be changed in the `CyfrinSolidityCert`, which is extending the `Soulbound` and it is expected to comply with `EIP-5484`:

```
 function setBaseURI(string memory baseURI) public onlyOwner {
    s_baseURI = baseURI;
    emit BaseURISet(baseURI);
}

function _baseURI() internal view override returns (string memory) {
    return s_baseURI;
}
```

And the `baseUri` is part of the metadata and can be changed after issuing.

Vulnerability Details

The following test can be added to `CyfrinSolidityCertTokenTest` file to verify that the `Issued` event is correctly emitted:

```
function testOnAttest_emitIssued() public asEAS {
    // Expect Transfer event
    vm.expectEmit(true, true, true, true);
    emit IERC721.Transfer(
        address(0),
        attestationRecipient1,
        uint256(UID_EXAMPLE)
    );

    // Expect CertificateMinted event
    vm.expectEmit(true, true, true, true);
    emit ICyfrinSolidityCert.CertificateMinted(
        uint256(UID_EXAMPLE),
        attestationRecipient1,
        100,
        EXPIRATION_TIME
    );

    // Expect MetadataUpdate event
    vm.expectEmit(true, true, true, true);
    emit IERC4906.MetadataUpdate(uint256(UID_EXAMPLE));

    // Expect Issued event (missing in current implementation)
    vm.expectEmit(true, true, true, true);
    emit IERC5484.Issued(
        ATTESTER_ADDRESS,
        attestationRecipient1,
        1,
        IERC5484.BurnAuth.IssuerOnly
    );

    assertTrue(
        s_target.attest(
            _attestation(
                SCHEMA_EXAMPLE,
                UID_EXAMPLE,
                attestationRecipient1,
                EXPIRATION_TIME,
                abi.encode(100),
                ATTESTER_ADDRESS
            )
        )
    );

    // Verify that the certificate was minted to the recipient
    assertEq(s_target.balanceOf(attestationRecipient1), 1);

    // Verify correct owner, score, and expiry time
    assertEq(s_target.ownerOf(uint256(UID_EXAMPLE)),
   attestationRecipient1);
    CyfrinSolidityCert.Certificate memory cert = s_target.certificate(
   uint256(UID_EXAMPLE));
    assertEq(cert.score, 100);
    assertEq(cert.expiryTime, EXPIRATION_TIME);
}
```

Impact

The contract does not fully comply with `EIP-5484` standard, as it fails to emit the required `Issued` event upon token issuance and also it can change the metadata after token issuance. This omission may cause compatibility issues with third-party applications relying on this event for tracking `Soulbound` token issuance.

Tools Used

Manual Review

Recommendations

Since `Soulbound` tokens are currently minted during `onAttest`, consider:

1. Extending the `_safeMint` function to include the `Issued` event emission.
2. Modifying `CyfrinSolidityCert` :: `onAttest` to explicitly emit the `Issued` event when a token is issued.

This will ensure full compliance with `EIP-5484` and improve interoperability with third-party tools.

Consider removing the change of the `baseUri` and this will fix the violation of the rule for metadata change.

Low

Low 01 Changing Attesters Renders Old Attestations Non-revokable

Summary

The `CyfrinSolidityCert` contract currently uses a single `allowedAttester` address, which can cause issues if the attester is changed, as old attestations would not be revokable.

Vulnerability Details

The `CyfrinSolidityCert` contract uses a single `allowedAttester` address to validate attestations and revocations. If the `allowedAttester` address is changed, attestations made by the previous attester would no longer be valid for revocation. This is because the `EAS` allows revocations only by the original attester but they will not be able to call `revoke` due to the `onlyAllowedAttester` modifier.

Impact

Changing the `allowedAttester` address would render all previous attestations non-revokable, which could lead to a situation where invalid or outdated attestations cannot be revoked. This could undermine the integrity of the attestation system and lead to potential misuse or security issues.

Tools Used

Manual Review

Recommendations

Consider changing the implementation to use a mapping of allowed attesters instead of a single address. This would allow multiple attesters to be valid at the same time and ensure that old attestations can still be revoked even if the `allowedAttester` is changed.

For example:

```
mapping(address => bool) private s_allowedAttesters;

modifier onlyAllowedAttester(address att) {
    require(s_allowedAttesters[att], InvalidAttester(att));
    _;
}

function setAllowedAttester(address allowedAttester, bool status)
   public onlyOwner {
    require(allowedAttester ≠ address(0), MustNotBeZeroAddress());
    s_allowedAttesters[allowedAttester] = status;
    emit AllowedAttesterSet(allowedAttester, status);
}
```

Low 02 Revoking Valid Attestations

Summary

The `CyfrinSolidityCert` contract is missing validation to check if an attestation is expired before allowing it to be revoked. Revoking expired attestations should not be allowed for the attester, and this action should potentially be restricted to the owner.

Vulnerability Details

In the `onRevoke` function, there is no check to ensure that the attestation being revoked is expired. This means that an attester can revoke attestations even before they have expired. Typically, valid attestations should not be revokable by the attester, and such actions should be restricted to the owner.

Impact

Allowing attestations to be revoked before they have expired can lead to inconsistencies and potential misuse. For example, an attester could revoke a valid attestation and destroy a user's certificate even though it should be valid. This could undermine the integrity of the attestation system and lead to trust loss.

Tools Used

Manual Review

Recommendations

Add a check in the `onRevoke` function to ensure that the attestation is expired before allowing it to be revoked.

For example:

```
require(block.timestamp > attestation.expirationTime, "Attestation is
    not expired and cannot be revoked");
```

Low 03 ERC-5192 Non Compliant

Summary

The `CyfrinSolidityCert` contract is not fully compliant with the ERC-5192 standard but it should be as it inherits the `Soulbound` contract. Specifically, it does not emit the `Locked` event when a token is minted and its status is locked.

Vulnerability Details

ERC-5192 is a standard for soulbound tokens, which are non-transferable tokens. According to the standard, when a token is minted and its status is locked, the `Locked` event SHOULD be emitted. The current implementation of the

`CyfrinSolidityCert` contract mints tokens without emitting this event, which makes it non-compliant with the ERC-5192 standard.

Impact

Non-compliance with the ERC-5192 standard can lead to interoperability issues with other systems and contracts that expect the `Locked` event to be emitted. This can result in unexpected behavior or incompatibility with platforms that rely on this event to track the status of soulbound tokens.

Tools Used

Manual Review

Recommendations

Modify the `onAttest` function to emit the `Locked` event when a token is minted and its status is locked.

For example:

```
emit Locked(tokenId);
```

Low 04 No Revocable Check

Summary

The `CyfrinSolidityCert` contract is missing validation to check if an attestation is revocable before allowing it to be attested.

Vulnerability Details

In the `onAttest` function, there is no check to ensure that the attestation being attested is marked as revocable. The `Attestation` struct includes a `revocable` boolean field that indicates whether an attestation can be revoked. However, the current implementation does not validate this field before performing an attestation.

```
struct Attestation {
    bytes32 uid; // A unique identifier of the attestation.
    bytes32 schema; // The unique identifier of the schema.
    uint64 time; // The time when the attestation was created (Unix
   timestamp).
    uint64 expirationTime; // The time when the attestation expires (
   Unix timestamp).
    uint64 revocationTime; // The time when the attestation was
   revoked (Unix timestamp).
    bytes32 refUID; // The UID of the related attestation.
    address recipient; // The recipient of the attestation.
    address attester; // The attester/sender of the attestation.
    bool revocable; // Whether the attestation is revocable.
    bytes data; // Custom attestation data.
}
```

Impact

Allowing attestations that are not marked as revocable can lead to unintended behavior and potential misuse. Such attestations cannot be revoked in the future even by the attester.

Tools Used

Manual Review

Recommendations

Add a check in the `onAttest` function to ensure that the attestation is marked as revocable before allowing it to be attested or else revoking it will be impossible.

For example:

```
function onAttest(Attestation calldata attestation, uint256 /*value*/
   )
    internal
    override
    whenNotPaused
    onlyAllowedAttester(attestation.attester)
    returns (bool)
{
     ...
+   require(attestation.revocable, "Attestation is not revocable");
```

```
    ...
}
```

Low 05 Attest & Revoke Will Not Refund On Some Chains Is Some Cases

Summary

The `CyfrinAttester`'s `attest` and `revoke` functionalities will not refund `msg.value` on some of the target chains (Arbitrum One, Base, Optimism and Ethereum). This is due to the fact that the version of the `EAS` contract deployed on these chains has different implementation of the `_resolveAttestation` function.

Vulnerability Details

The `CyfrinAttester`'s `attest` and `revoke` functionalities will not refund `msg.value` on some of the target chains (Arbitrum One, Base, Optimism and Ethereum). This is due to the fact that the version of the `EAS` contract deployed on these chains has different implementation of the `_resolveAttestation` function.

Specifically in the case in which the `resolver` is the zero address:

```
if (address(resolver) == address(0)) {
    // Ensure that we don't accept payments if there is no resolver.
    for (uint256 i = 0; i < length; ) {
        if (values[i] != 0) {
            revert NotPayable();
        }

        unchecked {
            ++i;
        }
    }

    return 0;
}
```

In the case in which the `values` array does not cause a revert the funds sent will not be refunded. This is fixed in the newer versions of the `EAS` contract:

```
if (address(resolver) == address(0)) {
    // Ensure that we don't accept payments if there is no resolver.
    if (value != 0) {
        revert NotPayable();
    }

    if (last) {
        _refund(availableValue);
    }

    return 0;
}
```

You can find the deployed `EAS` contract versions here

Impact

The inability of `EAS` to refund in such case is not well documented and can cause confusion to the attester expecting a refund. This will lead to loss of funds for the attester.

Tools Used

Manual Review

Recommendations

Either rethink the supported/target chains or document this well so attesters and users are aware of it.

Low 06 EIP-7572 Is A Draft Version

Summary

The `CyfrinSolidityCert` contract implements the EIP-7572 interface, which is based on EIP-7572. However, EIP-7572 is currently in draft status and is not finalized. Using an EIP in draft status poses risks, as the specification may change, leading to potential incompatibilities or vulnerabilities in the future.

Vulnerability Details

The contract implements `IERC7572`, which is derived from a draft proposal. Draft EIPs are subject to change, and their implementation in production contracts can lead to issues if the final specification differs significantly from the current draft.

Impact

If EIP-7572 changes, this contract may not conform to the final standard, requiring updates or even a full migration.

Tools Used

Manual Review

Recommendations

Wait until EIP-7572 reaches its final version before implementing it in live contracts.

Informational

Info 01 No Support Of Attestation Without Expiry

Summary

The Cyfrin Attester currently does not support attestations without expiry, although EAS supports such attestations.

Vulnerability Details

The `onAttest` function in the `CyfrinSolidityCert` contract requires that the `attestation.expirationTime` is greater than the current block timestamp. This means that attestations without an expiry time cannot be processed by the Cyfrin Attester.

However, the EAS supports attestations without expiry:

```
if (request.expirationTime != NO_EXPIRATION_TIME && request.
    expirationTime <= _time()) {
     revert InvalidExpirationTime();
}
```

Impact

This limitation restricts the flexibility of the attestation system, preventing the issuance of attestations that are meant to be valid indefinitely. This could be a significant limitation for use cases where permanent attestations are required.

Tools Used

Manual Review

Recommendations

Consider adding support for attestations without expiry by allowing `attestation.expirationTime` to be zero. This would align the Cyfrin Attester's functionality with the capabilities of EAS.

```
function onAttest(Attestation calldata attestation, uint256 /*value*/
    )
     internal
     override
     whenNotPaused
     onlyAllowedAttester(attestation.attester)
     returns (bool)
{
     // Obtain the tokenId from the Attestation uid, and mint the token
      to the recipient
     uint256 tokenId = uint256(attestation.uid);
     _safeMint(attestation.recipient, tokenId);

     // Set the expiry time of the token
     uint256 score = abi.decode(attestation.data, (uint256));
     require(score >= MIN_SCORE && score <= MAX_SCORE, InvalidScore(
    MIN_SCORE, MAX_SCORE, score));
-    require(attestation.expirationTime > block.timestamp,
    InvalidExpiryTime(attestation.expirationTime));
+    require(attestation.expirationTime == 0 || attestation.
    expirationTime > block.timestamp, InvalidExpiryTime(attestation.
    expirationTime));
     s_certificates[tokenId] = Certificate({score: uint128(score),
    expiryTime: uint128(attestation.expirationTime)});

     emit CertificateMinted(tokenId, attestation.recipient, score,
    attestation.expirationTime);
     emit MetadataUpdate(tokenId);

     // Return true to indicate that the attestation was successful
     return true;
}
```

Info 02 Configurable Min Score

Summary

The `MIN_SCORE` constant in the `CyfrinSolidityCert` contract is currently hardcoded and not configurable.

Vulnerability Details

The `MIN_SCORE` constant is set to 70, which enforces a minimum score for the certificates. This value is hardcoded and cannot be changed without modifying the contract code and redeploying it. This lack of configurability can be restrictive and may not accommodate future changes in requirements.

Impact

The inability to configure the minimum score dynamically limits the flexibility of the contract. If the minimum score needs to be adjusted, the contract would need to be redeployed, which is both time-consuming and costly. This could hinder the adaptability of the system to evolving standards or requirements.

Tools Used

Manual Review

Recommendations

Refactor the contract to make the `MIN_SCORE` value configurable. This can be achieved by replacing the constant with a state variable and providing a function to update its value. Ensure that only authorized users (e.g., the contract owner) can modify this value to maintain security.

Info 03 Missing Address Zero Checks

Summary

The `CyfrinAttester` contract is missing checks for `address(0)` for `admin` and `attester` in the constructor which can lead to potential issues if invalid addresses are provided.

Vulnerability Details

In the constructor of `CyfrinAttester`, there are no checks to ensure that the provided addresses (such as `admin`, `attester`) are not the zero address (`address(0)`). This can lead to the contract being initialized with invalid addresses, which can cause unexpected behavior.

Impact

If an invalid address (`address(0)`) is used for critical roles, it can lead to loss of control over the contract's functionality. For example, if the `admin` address is set to `address(0)`, it would be impossible to perform administrative functions such as pausing, unpausing, or withdrawing funds. Similarly, if the `attester` is set to `address(0)`, no attestations can be made.

Tools Used

Manual Review

Recommendations

Add checks in the constructor of `CyfrinAttester` to ensure that the provided addresses are not the zero address (`address(0)`).

For example:

```
require(admin ≠ address(0), "Admin address must not be zero");
require(attester ≠ address(0), "Attester address must not be zero");
```

Info 04 Keeper For Expired Attestation Revocation

Summary

The `CyfrinAttester` contract relies on the attester to manually call the `revoke` function to revoke expired attestations. This process could be automated using a keeper to ensure timely revocation of expired attestations.

Vulnerability Details

Currently, the revoke functionality in the `CyfrinAttester` contract depends on the attester to manually call it to revoke attestations. If the attester does not call this function in a timely manner, expired attestations may remain active longer than intended. This manual process can lead to delays and inconsistencies in the revocation of expired attestations.

Impact

The reliance on manual revocation by the attester can result in expired attestations remaining valid for longer periods. Automated revocation would ensure that expired attestations are promptly and consistently revoked, maintaining the integrity of the attestation system.

Tools Used

Manual Review

Recommendations

Consider implementing a keeper mechanism to automate the revocation of expired attestations. A keeper can periodically check for expired attestations and call the `revoke` function to ensure they are promptly revoked. This would reduce the reliance on manual intervention and improve the reliability and security of the attestation system.