OpenZeppelin | security

Zama Confidential Contracts 0.3.0 Release Audit

November 25, 2025

Table of Contents

Summary

Type	Library
Timeline	From 2025-10-20 To 2025-10-31
Languages	Solidity

Total Issues	13 (12 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	2 (2 resolved)
Notes & Additional Information	10 (9 resolved)

Scope

The audit scope covers changes in the OpenZeppelin Confidential Contracts repository, specifically the differential (diff) between base commit aed3fa1 (v0.2.0) and head commit 04342a7 (release-v0.3). This diff primarily introduces several extension contracts based on the `ERC7984` token contract.

In scope were the following files:

```
contracts/
├── finance/
│   ├── VestingWalletConfidential.sol
│   └── VestingWalletConfidentialFactory.sol
├── interfaces/
│   ├── IERC7984.sol
│   ├── IERC7984Receiver.sol
│   └── IERC7984Rwa.sol
├── token/
│   └── ERC7984/
│       ├── ERC7984.sol
│       ├── extensions/
│       │   ├── ERC7984ERC20Wrapper.sol
│       │   ├── ERC7984Freezable.sol
│       │   ├── ERC7984ObserverAccess.sol
│       │   ├── ERC7984Omnibus.sol
│       │   ├── ERC7984Restricted.sol
│       │   ├── ERC7984Rwa.sol
│       │   └── ERC7984Votes.sol
│       └── utils/
│           └── ERC7984Utils.sol
└── utils/
    ├── FHESafeMath.sol
    └── structs/
        └── temporary-Checkpoints.sol
```

After the fix review phase, the contracts were modified with few additional changes. All resolutions and the final state of the audited codebase mentioned in this report are contained at the commit `b55d406`.

System Overview

The updates to OpenZeppelin's Confidential Contracts, designed for use within the Zama fhEVM environment, primarily introduce ERC-7984 extensions, including `ERC7984Rwa`, `ERC7984Restricted`, `ERC7984Freezable`, `ERC7984ObserverAccess`, and `ERC7984Omnibus`:

- `ERC7984Restricted` implements a "blocklist" (which can be inverted to an "allowlist") for an ERC-7984-compliant confidential token.
- `ERC7984Freezable` allows for freezing an account's confidential assets up to a specified confidential amount.
- `ERC7984ObserverAccess` grants a designated "observer" view access to a user's confidential balances and transfer amounts.
- `ERC7984Omnibus` facilitates transfers between "omnibus" accounts using encrypted sub-account addresses. This conceals account addresses during transfers, at the cost of requiring external tracking of sub-account balances. It is particularly useful for transfers between exchanges using ERC-7984 tokens.
- `ERC7984Rwa` defines a confidential, fungible real-world asset (RWA) token with pausability, account freezing, blocklisting or allowlisting, and capabilities for "agents" to mint, burn, or force-transfer assets. These features support token management and regulatory compliance for RWA issuers. It builds on `ERC7984Freezable` and `ERC7984Restricted`.

In addition, several existing files related to `ConfidentialFungibleToken` were renamed to align with `ERC7984`, including `IConfidentialFungibleToken`, `ConfidentialFungibleToken`, and `ConfidentialFungibleTokenUtils`.

Finally, the `FHESafeMath` library was extended with `tryAdd` and `trySub` functions, which return a success boolean alongside the result (or 0 on failure).

Trust Assumptions and Privileged Roles

This section presents all the trust assumptions and privileged roles identified during the audit.

Developers extending or modifying these contracts must adhere to ACL allowance rules in the fhEVM ecosystem. All contracts or EOAs interacting with encrypted handles require explicit access grants; otherwise, functionality may fail, particularly when overriding logic in ERC7984 extensions.

In `ERC7984Restricted`, `transferFrom` operations that ensure the `from` and `to` addresses are not blocked, but do not validate `msg.sender` (the operator). As such, developers should review their compliance requirements and override the logic if necessary. For reference, USDT only validates `from`, while USDC validates `from`, `to`, and `msg.sender`.

The `discloseEncryptedAmount` function in `ERC7984` does not store or verify the decryption request ID in `finalizeDiscloseEncryptedAmount` to protect against replay attacks. As this is a user-initiated disclosure, it poses no direct threat, but off-chain event watchers must handle potential duplicate or out-of-order `AmountDisclosed` events.

Privileged Roles

Throughout the codebase, the following privileged roles were identified:

- In `ERC7984Rwa`, the "agent" role is trusted to responsibly:
  - pause and unpause token balance updates
  - add or remove users from the blocklist or allowlist (per configuration)
  - set confidential frozen amounts for accounts
  - confidentially mint or burn tokens for accounts
  - force-transfer confidential tokens between accounts
- The "admin" role can grant or revoke "agent" status.

Medium Severity

M-01 Agent Role Lacks Allowance in the ERC7984Rwa Contract

In the `ERC7984Rwa` contract, the agent role can mint, burn, or force the transfer of tokens between accounts. However, the agent is not granted an allowance for the encrypted amount of actual minted, burned, or transferred tokens. This prevents the agent from determining whether the operation has failed, for example, if the `_mint` function returns an encrypted `0`.

Consider granting allowances on the return values of the `_mint`, `_burn`, and `_forceUpdate` functions to the agent (`msg.sender`).

Update: *Resolved in pull request #242 at commit aefa22f.*

Low Severity

L-01 `tryDecrease` Returns `(true, uninitialized)` when `oldValue` is uninitialized and `delta` Is Initialized Zero

The `FHESafeMath` library contains two pairs of operations: `tryIncrease/tryDecrease` and `tryAdd/trySub`. According to the comments, the difference between them is that when the operation fails, `tryIncrease/tryDecrease` will return the original `oldValue` value, while `tryAdd/trySub` will return `0`. For example, the `tryDecrease` will return `false, uninitialized` when `oldValue` is uninitialized and `delta` is `1`.

However, when `oldValue` is uninitialized and delta is initialized `0`, `tryDecrease` succeeds and returns an uninitialized result. Besides being unexpected for users, this also brings an inconsistency for `tryIncrease/tryDecrease`. Let's say the `oldValue` (a) is uninitialized, and the `delta` (b) is 0:

- `tryIncrease`: a + b = `(true, 0)`
- `tryDecrease`: a - b = `(true, uninitialized)`

In contrast, the behavior is consistent for `tryAdd/trySub` - the results are both `(true, 0)`.

Consider changing the logic of these operations so that **only the initialized value (e.g., `0`) will be returned when the `success` is true**, which aligns more with the `FHE` library (treat uninitialized as `0`). Alternatively, consider removing `return (FHE.eq(oldValue, delta), oldValue);` from `tryDecrease`. So, when `oldValue` is uninitialized:

- if `delta` is also uninitialized, return `(true, uninitialized)`
- if `delta == 0`, return `(true, 0)`
- if `delta > 0`, return `false, uninitialized`

Update: *Resolved in pull request #241 at commit 9005e9b. The lib will treat an uninitialized value as 0, and only return an uninitialized value when both parameters are uninitialized.*

L-02 Missing Allowance Grant in `confidentialAvailable`

The `ERC7984Rwa` contract includes a public `confidentialAvailable` function that accepts an `address account` parameter and returns the confidential available (unfrozen) balance for an account. However, the allowance for the encrypted available balance has not been granted, which prevents the use of this public function's return value.

If it is intended to leave this responsibility to users, either by overriding the function or using an access management contract, then consider clearly documenting this aspect. Otherwise, consider granting the allowance on the available balance to `account`.

Update: *Resolved in pull request #252 at commit b4135ca.*

Notes & Additional Information

N-01 Misleading Function Name of `unblockUser`

The `unblockUser` function is documented as "Unblock a user account" and the current implementation invokes `_allowUser`, which sets the restriction to `Restriction.ALLOWED`. This behavior is appropriate because users can override the `isUserAllowed` method to implement an allowlist rather than relying on the default blocklist semantics. Nevertheless, the function name `unblockUser` is misleading, as it does not reset the restriction to `Restriction.DEFAULT`.

Consider changing the function name to `allowUser`.

Update: *Resolved in pull request #244 at commit e51154b. The override to allowlist documentation is removed in pull request #245, and `UnblockUser` invokes `_resetUser`. The team stated:*

> *If a user changes the logic of `isUserAllowed`, then they should also change the logic of `blockUser` and `unblockUser`.*

N-02 Inconsistency in the Sequence of Allowance Grant and Transfer Operations

In the `_confidentialTransferFromOmnibus` function, the allowance grant operations are invoked before the `confidentialTransferFrom` call. However, in the `_confidentialTransferFromAndCallOmnibus` function, allowances on `sender` and `recipient` are granted after `confidentialTransferFromAndCall` returns, i.e., after the receiver callback finishes. If the callback logic checks these allowances or relies on these allowances, it could fail or revert.

Consider moving the allowance grant on `sender` and `recipient` before the `confidentialTransferFromAndCall` call in the `_confidentialTransferFromAndCallOmnibus` function.

Update: *Resolved in pull request #250 at commit 6d3c525.*

N-03 Unused Import

The import `import {FHESafeMath} from "./../../../utils/FHESafeMath.sol";` in `ERC7984Rwa.sol` is unused and can be removed.

Consider removing unused imports to improve the overall clarity and maintainability of the codebase.

Update: *Resolved in pull request #243 at commit c911648.*

N-04 Missing Named Parameters in Mapping

Since Solidity 0.8.18, mappings can include named parameters to provide more clarity about their purpose. Named parameters allow mappings to be declared in the form `mapping(KeyType KeyName? => ValueType ValueName?)`. This feature enhances code readability and maintainability.

In the `_observers` state variable of the `ERC7984ObserverAccess` contract, the mapping does not have any named parameters.

Consider adding named parameters to mappings in order to improve the readability and maintainability of the codebase.

Update: *Resolved in pull request #251 at commit 4cfe803.*

N-05 Missing Security Contact

Providing a specific security contact (such as an email address or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, multiple instances of contracts not having a security contact were identified:

- The `VestingWalletConfidential` abstract contract

- The `VestingWalletConfidentialFactory` abstract contract
- The `ERC7984` abstract contract
- The `ERC7984ERC20Wrapper` abstract contract
- The `ERC7984Freezable` abstract contract
- The `ERC7984ObserverAccess` abstract contract
- The `ERC7984Omnibus` abstract contract
- The `ERC7984Restricted` abstract contract
- The `ERC7984Rwa` abstract contract
- The `ERC7984Votes` abstract contract
- The `ERC7984Utils` library
- The `FHESafeMath` library

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Resolved. The team lists a specific `SECURITY.md` file under the repository, which contains comprehensive security information, including the security contact.*

N-06 Variable Initialized With Default Value

In `VestingWalletConfidentialFactory.sol`, the `i` variable is initialized with the default value `0`.

To avoid wasting gas, consider not initializing variables with their default values.

Update: *Acknowledged, not resolved. The team stated:*

> *This follows the pattern defined in other OpenZeppelin repos. The optimizer will ensure that excess gas is not used here.*

N-07 Constants Not Using `UPPER_CASE` Format

The `VestingWalletStorageLocation`, `VestingWalletCliffStorageLocation`, and `ERC7821WithExecutorStorageLocation` constants are not declared using the `UPPER_CASE` format.

According to the Solidity Style Guide, constants should be named with all capital letters with underscores separating words. For better readability, consider following this convention.

Update: *Resolved in pull request #247 at commit cfeba31.*

N-08 Missing Docstrings

Within `ERC7984Rwa.sol`, in the public `AGENT_ROLE` state variable, the docstring is missing.

Consider thoroughly documenting all states that are part of the contract's public API.

Update: *Resolved in pull request #249 at commit af998d7.*

N-09 Misleading Docstrings

Within the `ERC7984Rwa` contract, multiple instances of misleading docstrings were identified:

- The docstrings for the two versions of the `setConfidentialFrozen` function appear to be reversed.
- The comments above the `forceConfidentialTransferFrom` public function merely state that the transfer is forced by skipping compliance checks. However, the checks on frozen assets and the recipient are preserved.

Consider swapping the docstrings in line 100 and line 109, and updating the `forceConfidentialTransferFrom` function comment to clarify which compliance check is skipped.

Update: *Resolved in pull request #249 at commit af998d7 and pull request #255 at commit 19dd8e0.*

N-10 Misdocumented Revert Error in `ERC7984Restricted._checkRestriction`

The `ERC7984Restricted` extension enforces account-level transfer restrictions via `_checkRestriction`. Within `ERC7984Restricted.sol`, in line 94, the NatSpec for `_checkRestriction` states “Reverts with {ERC20Restricted}”, while the contract defines and uses the custom error `UserRestricted(address)` for restriction violations. No symbol named `ERC20Restricted` exists in the codebase. This error in the NatSpec can cause off-chain or integration code that decodes a revert based on the documented `{ERC20Restricted}` name to fail to match the emitted `UserRestricted(address)` selector.

Consider aligning the documentation and implementation. Either update the NatSpec to "Reverts with {UserRestricted}", or rename the error to `ERC20Restricted(address)` to match the current documentation.

Update: *Resolved in pull request #245 at commit cda1ca7.*

Conclusion

Overall, the codebase was found to be thoughtfully designed, tested, and consistent with previously audited components. One medium-severity findings were also reported, related to lack of allowance for agent role to mint, burn or fornce the transfer of tokens. The rest of the findings were low-severity or notes, predominantly documentation and ergonomics, small robustness improvements, and minor consistency cleanups.

The OpenZeppelin smart contract development team is highly appreciated for their collaboration and responsiveness throughout the audit.