BLOCKSEC

Security Audit
Report for RWA EVM Contracts

Date: April 15, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Bitget
Target	RWA EVM Contracts

Version History

Version	Date	Description
1.0	April 15, 2026	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-04-15 09:24:25+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About the Audit Target

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of RWA EVM Contracts of Bitget.

The protocol introduces an on-chain Real World Asset (RWA) tokenization implementation on EVM-compatible chains, enabling regulated issuance and redemption of asset-backed tokens representing securities. The system is organized around three core contracts: `RWAToken`, `RWATokenManager`, and `RWATokenFactory`. `RWAToken` is an upgradeable ERC-20 token deployed via a `BeaconProxy` pattern, implementing a rebase mechanism in which all token balances are stored as raw values and divided by a global rebase index at read time. The rebase index is updated through a split ratio, allowing the protocol to model corporate splits and reverse splits by adjusting the index proportionally across all holders simultaneously without modifying individual balances. Each `RWAToken` instance also enforces three-layered compliance guards: KYC verification through `IDRegistry`, freeze controls, and protocol-wide sanctions screening via `SanctionsList`. `RWATokenFactory` handles token deployment and wires each instance to shared infrastructure, including a `CorporateActionOracle`, which governs time-windowed corporate actions such as rebase splits and name and symbol updates. `RWATokenManager` acts as the primary entry point for user-facing mint and redeem flows, both of which operate through an EIP-712 Attestation signed by a trusted `SIGNER_ROLE`. Each attestation binds the token, quantity, price, settlement amount, and expiration into a single one-time-use authorization. Price integrity is enforced by `SanityCheckOracle`, which validates the attested price against an independent reference via either a Pyth Lazer Pro on-chain verification or a `SignedPrice` signed by multi-party, rejecting deviations exceeding a configurable basis-point threshold. Throughput is governed by `RateLimiter`, which applies a linear-decay sliding-window quota at both the global per-token and per-user levels. Market session eligibility is enforced by MarketHours. The protocol is designed for a KYC DEX intermediary model, where partner DEX contracts hold calling rights and are responsible for end-user compliance off-chain before invoking the Manager on behalf of their users.

Note this audit only focuses on the smart contracts in the following directories/files:

- src

Other files are not within the scope of the audit. Additionally, all dependencies of the Target are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

[1] https://github.com/blcokchainbg/rwa-evm

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (Version 1), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (Version 0), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
rwa-evm	Version 1	eb2f9cdd26ce4f745e0dafe02e993195c5d7c7ef
	Version 2	0d09252e034e883dc67ea540705076360fdafbc4

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the Target, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of the Target.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control

- Permission management
- Whitelist and blacklist mechanisms
- Initialization consistency
- Improper use of the proxy system
- Reentrancy
- Denial of Service (DoS)
- Untrusted external call and control flow
- Exception handling
- Data handling and flow
- Events operation
- Error-prone randomness
- Oracle security
- Business logic correctness
- Semantic and functional consistency
- Emergency mechanism
- Economic and incentive impact

1.3.2 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

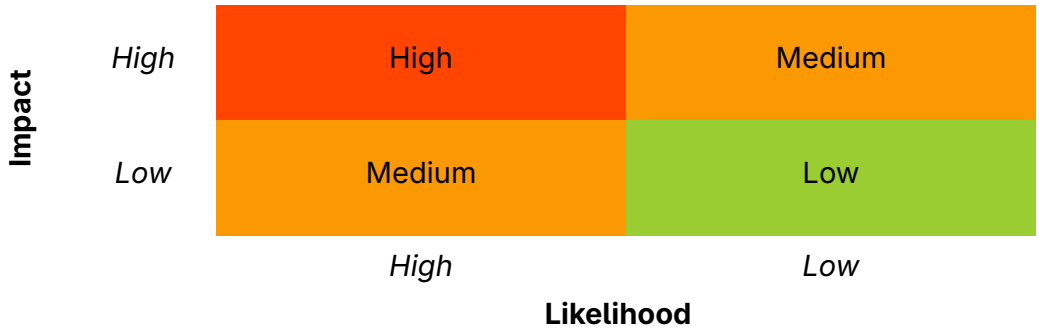
Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Table 1.1: Vulnerability Severity Classification



- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we found **four** potential security issues. Besides, we have **five** recommendations and **thirteen** notes.

- Low Risk: 4
- Recommendation: 5
- Note: 13

ID	Severity	Description	Category	Status
1	Low	Incompatible event interfaces	Security Issue	Fixed
2	Low	Incorrect rounding direction in rebase index updates	Security Issue	Confirmed
3	Low	Inconsistent price scaling	Security Issue	Fixed
4	Low	Incorrect staleness validation for Pyth price feeds	Security Issue	Confirmed
5	-	Fix typos	Recommendation	Fixed
6	-	Fix incorrect events	Recommendation	Fixed
7	-	Align the documentation with the current implementation	Recommendation	Fixed
8	-	Strengthen length validation in function `validatePrice()`	Recommendation	Fixed
9	-	Fix the payload format for renaming actions	Recommendation	Fixed
10	-	Unify configurations across `RWAToken` instances	Note	-
11	-	Attestation issuance responsibility	Note	-
12	-	Price parity assumption for `settleToken`	Note	-
13	-	Divergent compliance paths across transfer flows	Note	-
14	-	Ensure enough funds to pay users in `redeemVault`	Note	-
15	-	Dust balance due to precision loss	Note	-
16	-	Potential centralization risks	Note	-
17	-	Off-chain calculation for `settleAmount`	Note	-
18	-	Asset backing assumption for inventory management functions	Note	-
19	-	Off-chain handling after delisting	Note	-
20	-	Fixed decimal assumption in price normalization	Note	-
21	-	Allowance usage during rebasing process	Note	-
22	-	Front-run risks of attestation submission	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Incompatible event interfaces

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `RWAToken` inherits from `IRWAToken` and `RWATokenRebaseUpgradeable`, and should maintain consistency between the interface and implementation. However, the event definitions are inconsistent. Specifically, interface `IRWAToken` declares events `NameChanged`, `SymbolChanged`, and `Rebased` as the public event surface for renaming and rebase-related corporate actions. The corresponding functions in contract `RWATokenRebaseUpgradeable` emit `NameUpdated`, `SymbolUpdated`, and `RebaseBySplit`, instead.

```
23   event NameChanged(string oldName, string newName);
24   event SymbolChanged(string oldSymbol, string newSymbol);
25   event SanctionsListSet(address indexed oldSanctionsList, address indexed newSanctionsList);
26   event CorporateActionOracleSet(address indexed oldActionOracle, address indexed
         newActionOracle);
27   event MinterUpdated(address oldMinter, address newMinter);
28   event AccountFrozen(address indexed account);
29   event AccountUnfrozen(address indexed account);
30   event Rebased(uint256 indexed oldTotalStocks, uint256 indexed newTotalStocks, uint256
         splitFrom, uint256 splitTo);
```

Listing 2.1: src/interfaces/IRWAToken.sol

```
164  function _rebaseBySplit(
165      uint32 from,
166      uint32 to
167  ) internal virtual {
168      if (from == 0 || to == 0) revert InvalidIndex();
169      RebaseStorage storage $ = _getRebaseStorage();
170      uint256 oldIndex = $._rebaseIndex;
171      uint256 newIndex = oldIndex * from / to;
172      _rebase(newIndex);
173      emit RebaseBySplit(from, to, oldIndex, newIndex);
174  }
```

Listing 2.2: src/RWATokenRebaseUpgradeable.sol

```
269  function _setName(
270      string memory newName
271  ) internal virtual {
272      RebaseStorage storage $ = _getRebaseStorage();
273      string memory oldName = $._name;
```

```
274         $._name = newName;
275         emit NameUpdated(oldName, newName);
276     }
277
278     function _setSymbol(
279         string memory newSymbol
280     ) internal virtual {
281         RebaseStorage storage $ = _getRebaseStorage();
282         string memory oldSymbol = $._symbol;
283         $._symbol = newSymbol;
284         emit SymbolUpdated(oldSymbol, newSymbol);
285     }
```

Listing 2.3: src/RWATokenRebaseUpgradeable.sol

Impact Any integrators that subscribe to the interface `IRWAToken` events will not observe the corresponding on-chain actions, even though the state changes succeed.

Suggestion Unify the emitted event definitions on events `NameChanged`, `SymbolChanged`, and `Rebased`.

2.1.2 Incorrect rounding direction in rebase index updates

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In the contract `RWATokenRebaseUpgradeable`, the function `_rebaseBySplit()` computes the new index as `oldIndex * from / to`. However, the division uses floor rounding, resulting in a smaller calculated `rebaseIndex` than the actual value. As a result, both functions `balanceOf()` and `totalSupply()` may return a slightly larger value than intended, since the return values are derived using division by `rebaseIndex`.

```
164     function _rebaseBySplit(
165         uint32 from,
166         uint32 to
167     ) internal virtual {
168         if (from == 0 || to == 0) revert InvalidIndex();
169         RebaseStorage storage $ = _getRebaseStorage();
170         uint256 oldIndex = $._rebaseIndex;
171         uint256 newIndex = oldIndex * from / to;
172         _rebase(newIndex);
173         emit RebaseBySplit(from, to, oldIndex, newIndex);
174     }
175
176     /// @dev Set rebaseIndex directly. newIndex must be in (0, MAX_INDEX].
177     function _rebase(
178         uint256 newIndex
179     ) internal virtual {
180         if (newIndex == 0 || newIndex > MAX_INDEX) revert InvalidIndex();
181         RebaseStorage storage $ = _getRebaseStorage();
182         $._rebaseIndex = newIndex;
```

```
183     }
```

Listing 2.4: src/RWATokenRebaseUpgradeable.sol

```
81      function totalSupply() public view virtual override returns (uint256) {
82          RebaseStorage storage $ = _getRebaseStorage();
83          return $._totalRaw / $._rebaseIndex;
84      }
85
86      function balanceOf(
87          address account
88      ) public view virtual override returns (uint256) {
89          RebaseStorage storage $ = _getRebaseStorage();
90          return $._rawBalances[account] / $._rebaseIndex;
91      }
```

Listing 2.5: src/RWATokenRebaseUpgradeable.sol

Impact The functions `balanceOf()` and `totalSupply()` may return a greater value than the precise amount.

Suggestion Revise the code logic accordingly.

Feedback from the project The project team clarified that any resulting deviation in functions `balanceOf()` and `totalSupply()` is limited to the wei level and is considered acceptable under the current requirements. As the token is not intended for on-chain DeFi integrations at this stage, the risk of inflated balances is limited. If such use cases are supported in the future, a wrapper-based model similar to `wstETH` would likely be adopted.

2.1.3 Inconsistent price scaling

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `RWATokenManager`, functions `mintWithSignature()` and `redeemWithSignature()` compute `mintUsdValue` and `redemptionUsdValue` using the hardcoded constant `RWA_NORMALIZER`. In contrast, the contract `SanityCheckOracle` defines the attestation price decimals as a configurable parameter, `attestationPriceDecimals`. A mismatch between `RWA_NORMALIZER` and `attestationPriceDecimals` would cause the USD value computation in `RWATokenManager` to be inconsistent with the oracle's price normalization. As a result, the required settlement amounts would be miscomputed, potentially allowing incorrect settlement amounts or blocking legitimate mint and redeem operations.

```
93      function mintWithSignature(
94          Attestation calldata attestation,
95          bytes calldata signerSignature,
96          bytes calldata priceProof
97      ) external override nonReentrant whenNotPaused whenMintingNotPaused(attestation.rwaToken)
            returns (uint256) {
98          bytes32 userId = idRegistry.getRegisteredID(GM_IDENTIFIER, _msgSender());
```

```
 99        if (userId == bytes32(0)) revert UserNotRegistered();
100
101        _verifyAttestation(attestation, OrderSide.MINT, signerSignature, priceProof);
102        executedAttestationIds[attestation.attestationId] = true;
103
104        uint256 mintUsdValue = (attestation.quantity * attestation.price) / RWA_NORMALIZER;
105        if (mintUsdValue < minimumMintUSD) revert BelowMinimum();
106
107        uint256 decimals = IERC20Metadata(attestation.settleToken).decimals();
108        uint256 stableCoinAmount = _convertToStableCoinPrecision(mintUsdValue, decimals);
109        if (attestation.settleAmount < stableCoinAmount) revert InsufficientSettleAmount();
110
111        rateLimiter.checkAndUpdateRateLimit(attestation.side, attestation.rwaToken, userId,
               mintUsdValue);
```

Listing 2.6: src/RWATokenManager.sol

```
119    function redeemWithSignature(
120        Attestation calldata attestation,
121        bytes calldata signerSignature,
122        bytes calldata priceProof
123    ) external override nonReentrant whenNotPaused whenRedeemNotPaused(attestation.rwaToken)
           returns (uint256) {
124        bytes32 userId = idRegistry.getRegisteredID(GM_IDENTIFIER, _msgSender());
125        if (userId == bytes32(0)) revert UserNotRegistered();
126
127        _verifyAttestation(attestation, OrderSide.REDEEM, signerSignature, priceProof);
128        executedAttestationIds[attestation.attestationId] = true;
129
130        uint256 redemptionUsdValue = (attestation.quantity * attestation.price) / RWA_NORMALIZER;
131        if (redemptionUsdValue < minimumRedemptionUSD) revert BelowMinimum();
```

Listing 2.7: src/RWATokenManager.sol

```
31    uint8 public attestationPriceDecimals;
```

Listing 2.8: src/SanityCheckOracle.sol

Impact Settlement validation may be incorrect.

Suggestion Revise the code logic accordingly.

2.1.4 Incorrect staleness validation for Pyth price feeds

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In the contract `SanityCheckOracle`, the function `_validateViaPyth()` utilizes the variable `update.timestamp` to verify the freshness of price data. However, the update-level timestamp (`update.timestamp`) and the feed-level one `feedUpdateTimestamp` are different in Pyth Pro. For example, during market closures, the update timestamp may advance while the

actual price remains a carried-forward value from a previous session. As a result, the contract may incorrectly identify an obsolete price as fresh data.

```
150       // 3. Staleness check (Pyth timestamp is microseconds).
151       uint256 priceTimestampSec = uint256(update.timestamp) / MICROS_PER_SECOND;
152       if (block.timestamp > priceTimestampSec + maxStaleness) {
153           revert PythPriceStale(priceTimestampSec, maxStaleness);
154       }
```

Listing 2.9: src/SanityCheckOracle.sol

Impact The contract may accept obsolete carried-forward prices as valid during market closures, leading to potential arbitrage or incorrect settlement values.

Suggestion Use the `feedUpdateTimestamp` within the price payload to perform the staleness check.

Feedback from the project The project confirmed that attestations have a short validity period and there is no risk of execution across weekends.

2.2 Recommendation

2.2.1 Fix typos

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the project, there are some typos as follows:

1. In contracts `IRateLimiter` and `RateLimiter`, errors `InvaidLimit()` and `InvaidWindow()` misspell "Invalid".

2. In contract `RWATokenManager`, function `setMarketHours()` declares `oldMartketHoursAddr` with an extra `t`.

3. In contract `IDRegistry`, comments above `rwaRole` and `userIDs` misspell "rwatoken".

4. In contract `RWATokenRebaseUpgradeable`, the annotation for the function `rebaseBySplit()` contains a mistake regarding the split ratio explanation. It states that a `1-for-10 split` results in a 10x balance increase, whereas standard terminology defines this as a `10-for-1 split`.

5. The contract `RWATokenManager` defines the event `RateLimiterSet` with parameters `oldOndoRateLimiter` and `newOndoRateLimiter`, where the "Ondo" reference originates from an external project and causes confusion about ownership and rate limiter implementation.

6. In the contract `IDRegistry`, the function `getRegisteredID()` stores registration data under the `rwaToken` namespace. However, in the contract `RWATokenManager`, the function `mintWithSignature()` does not query the actual `attestation.rwaToken` and instead always uses `GM_IDENTIFIER`. This inconsistency may create ambiguity regarding the intended scope of KYC validation.

```
7   error InvaidLimit();
8   error InvaidWindow();
```

Listing 2.10: src/interfaces/IRateLimiter.sol

```
299   function setMarketHours(
300       address _marketHours
301   ) external onlyRole(CONFIGURER_ROLE) {
302       if (_marketHours == address(0)) revert ZeroAddress();
303       address oldMartketHoursAddr = address(marketHours);
304       if (_marketHours == oldMartketHoursAddr) revert AlreadySet();
305       marketHours = IMarketHours(_marketHours);
306       emit MarketHoursSet(oldMartketHoursAddr, _marketHours);
307   }
```

Listing 2.11: src/RWATokenManager.sol

```
11   //rwatoke ==> role
12   mapping(address => bytes32) public rwaRole;
13
14   //rwatoke ==> user address ==> user ID
15   mapping(address => mapping(address => bytes32)) public userIDs;
```

Listing 2.12: src/IDRegistry.sol

```
162   /// @dev Split/reverse-split. from:to ratio applied to current index.
163   ///      e.g. rebaseBySplit(1, 10) = 1-for-10 split, balances increase 10x.
```

Listing 2.13: src/RWATokenRebaseUpgradeable.sol

```
78   event RateLimiterSet(address indexed oldOndoRateLimiter, address indexed newOndoRateLimiter);
```

Listing 2.14: src/interfaces/IRWATokenManager.sol

```
7 *        This is our own compliance interface, separate from Ondo's,
```

Listing 2.15: src/interfaces/ICompliance.sol

```
93   function mintWithSignature(
94       Attestation calldata attestation,
95       bytes calldata signerSignature,
96       bytes calldata priceProof
97   ) external override nonReentrant whenNotPaused whenMintingNotPaused(attestation.rwaToken)
         returns (uint256) {
98       bytes32 userId = idRegistry.getRegisteredID(GM_IDENTIFIER, _msgSender());
99       if (userId == bytes32(0)) revert UserNotRegistered();
```

Listing 2.16: src/RWATokenManager.sol

```
20   //constant
21   address public constant GM_IDENTIFIER = address(uint160(uint256(keccak256(abi.encodePacked("
         global_markets")))));
```

Listing 2.17: src/RWATokenManager.sol

```
31   function getRegisteredID(
32       address rwaToken,
33       address user
```

```
34    ) external view override returns (bytes32 userID) {
35        if (rwaToken == address(0)) revert ZeroAddress();
36        if (user == address(0)) revert ZeroAddress();
37        return userIDs[rwaToken][user];
38    }
```

Listing 2.18: src/IDRegistry.sol

Suggestion Fix the typo errors.

2.2.2 Fix incorrect events

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the project, several contracts emit incorrect events or contain incorrect fields in the events.

1. In contract `RWATokenManager`, events `MinimumMintAmountSet` and `MinimumRedemptionAmountSet` track `oldAmount` as the updated amount rather than the previous amount.

2. In contract `RWATokenFactory`, both functions `updateOperator()` and `updateConfigurer()` emits events `AdminUpdated`. However, these two functions are used to update the state variables `operator` and `tokenManager`, not the `admin`.

3. In contract `SanityCheckOracle`, the function `_checkDeviation()` calculates the variable `deviationBps` by rounding down. However, the check for the maximum deviation is stricter. This results in a situation where a transaction reverts for exceeding the limit, but the error message displays a value that is valid. This inconsistency confuses debugging.

```
233   function setMinimumOrderUsd(
234       uint256 amount
235   ) external onlyRole(CONFIGURER_ROLE) {
236       if (minimumMintUSD == amount) revert AlreadySet();
237       minimumMintUSD = amount;
238       emit MinimumMintAmountSet(minimumMintUSD, amount);
239   }
240
241   function setMinimumRedemptionUsd(
242       uint256 amount
243   ) external onlyRole(CONFIGURER_ROLE) {
244       if (minimumRedemptionUSD == amount) revert AlreadySet();
245       minimumRedemptionUSD = amount;
246       emit MinimumRedemptionAmountSet(minimumRedemptionUSD, amount);
247   }
```

Listing 2.19: src/RWATokenManager.sol

```
59    function updateOperator(
60        address newOperator
61    ) external onlyRole(DEFAULT_ADMIN_ROLE) {
62        _zeroAddressCheck(newOperator);
63
64        _ensureChanged(operator, newOperator);
```

```
65        operator = newOperator;
66        emit AdminUpdated(newOperator);
67    }
68
69    function updateConfigurer(
70        address newConfigurer
71    ) external onlyRole(DEFAULT_ADMIN_ROLE) {
72        _zeroAddressCheck(newConfigurer);
73
74        _ensureChanged(configurer, newConfigurer);
75        configurer = newConfigurer;
76        emit AdminUpdated(newConfigurer);
77    }
```

Listing 2.20: src/RWATokenFactory.sol

```
214   function _checkDeviation(
215       uint256 priceA,
216       uint256 priceB,
217       uint16 _maxDeviationBps
218   ) internal pure returns (uint256 deviationBps) {
219       if (priceB == 0) revert ZeroPrice();
220
221       uint256 diff = priceA > priceB ? priceA - priceB : priceB - priceA;
222
223       deviationBps = (diff * BPS_DENOMINATOR) / priceB;
224
225       if (diff * BPS_DENOMINATOR > priceB * uint256(_maxDeviationBps)) {
226           revert PriceDeviationTooLarge(priceA, priceB, deviationBps);
227       }
228   }
```

Listing 2.21: src/SanityCheckOracle.sol

Suggestion Fix incorrect event names or fields.

2.2.3 Align the documentation with the current implementation

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The current documentation contains several inconsistencies between descriptions and implementations as follows:

1. In the document `README`, the statement "User submits attestation + signature + price proof to 'RWATokenManager" (line 58) is inconsistent with the current architecture. Under the actual design, users interact through the DEX rather than submitting these materials directly to the contract `RWATokenManager`.

2. The documentation `README` states that `OPERATOR_ROLE` can both pause and unpause the system (line 40). However, in the implementation, `OPERATOR_ROLE` can only invoke the function `pause()`, while the function `unpause()` is restricted to `DEFAULT_ADMIN_ROLE`.

3. The document `README` lists delisting as a corporate action (line 3), implying it should be verified via the contract `CorporateActionOracle`. However, the functions `delist()` and `relist()` of contract `RWAToken` can be executed directly by `CONFIGURER_ROLE` without validation through the contract `CorporateActionOracle`.

```
40| `OPERATOR_ROLE` | Pause/unpause, emergency withdraw, pause minting/redemptions | All contracts |
```

Listing 2.22: README.md

```
582. **User** submits attestation + signature + price proof to `RWATokenManager`
```

Listing 2.23: README.md

```
41    function pause() external onlyRole(OPERATOR_ROLE) {
42        _pause();
43    }
44
45    function unpause() external onlyRole(DEFAULT_ADMIN_ROLE) {
46        _unpause();
47    }
```

Listing 2.24: src/RWACommon.sol

```
174   function delist() external virtual whenNotPaused onlyRole(CONFIGURER_ROLE) {
175       RWATokenStorage storage $ = _getRWATokenStorage();
176       if ($.delisted) revert TokenDelisted();
177       $.delisted = true;
178       emit Delisted();
179   }
180
181   function relist() external virtual whenNotPaused onlyRole(CONFIGURER_ROLE) {
182       RWATokenStorage storage $ = _getRWATokenStorage();
183       if (!$.delisted) revert TokenNotDelisted();
184       $.delisted = false;
185       emit Relisted();
186   }
187
188   function balanceOf(
189       address account
190   ) public view virtual override returns (uint256) {
191       RWATokenStorage storage $ = _getRWATokenStorage();
192       if ($.delisted) return 0;
193       return super.balanceOf(account);
194   }
```

Listing 2.25: src/RWAToken.sol

```
3On-chain tokenization of real-world securities. Each token represents one share (1 Token = 1 Share
    ) with support for corporate actions such as stock splits, reverse splits, name changes, and
    delisting.
```

Listing 2.26: README.md

Suggestion Ensure that the documentation remains consistent with the implementation.

2.2.4 Strengthen length validation in function `validatePrice()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In contract `SanityCheckOracle`, function `validatePrice()` only checks that the variable `priceProof` is non-empty before processing proof data. Malformed or undersized inputs can therefore trigger unexpected reverts. It is recommended to add explicit length checks for each supported proof type.

```
67   function validatePrice(
68       address rwaToken,
69       uint256 attestationPrice,
70       bytes calldata priceProof
71   ) external whenNotPaused onlyRole(PRICE_CONSUMER_ROLE) {
72       if (priceProof.length == 0) revert InvalidProofData();
73
74       uint8 proofType = uint8(priceProof[0]);
75       bytes calldata proofData = priceProof[1:];
76
77       if (proofType == PROOF_TYPE_PYTH) {
78           _validateViaPyth(rwaToken, attestationPrice, proofData);
79       } else if (proofType == PROOF_TYPE_SIGNED) {
80           _validateViaSigned(rwaToken, attestationPrice, proofData);
81       } else {
82           revert InvalidProofType(proofType);
83       }
```

Listing 2.27: src/SanityCheckOracle.sol

Suggestion Add explicit proof-length validation for each supported proof type before decoding or processing the payload.

2.2.5 Fix the payload format for renaming actions

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `CorporateActionOracle`, function `_validatePayload()` contains uncommon decoding methods for the renaming action payload. Specifically, the payload is decoded in the order of `symbol` and `name`. This format is different from the usual order of `name` and `symbol` in ERC20 tokens, and may cause confusion to integrators.

```
151  function _validatePayload(
152      ActionType actionType,
153      bytes memory payload
154  ) internal pure {
155      if (payload.length == 0) revert PayloadEmpty();
156
157      if (actionType == ActionType.Rebase) {
158          if (payload.length != 64) revert InvalidPayload();
159          (uint32 splitFrom, uint32 splitTo) = abi.decode(payload, (uint32, uint32));
```

```
160             if (splitFrom == 0 || splitTo == 0) revert InvalidPayload();
161         } else if (actionType == ActionType.NewNameAndSymbol) {
162             (string memory symbol, string memory name) = abi.decode(payload, (string, string));
163             if (bytes(symbol).length == 0 || bytes(name).length == 0) revert InvalidPayload();
```

Listing 2.28: src/CorporateActionOracle.sol

Suggestion Fix the order of the payload format.

2.3 Note

2.3.1 Unify configurations across `RWAToken` instances

Introduced by `Version 1`

Description Although contract `RWAToken` uses the Beacon proxy pattern to unify implementation logic across instances, updates to configuration in contract `RWATokenFactory` may create inconsistencies between already deployed tokens and future deployments. If the protocol intends to maintain a consistent privileged-control model across all `RWAToken` instances, it should ensure that a shared configuration mechanism, governed by `DEFAULT_ADMIN_ROLE`, is in place and functions correctly to synchronize settings across both existing and newly deployed tokens.

```
109     function createToken(
110         string calldata name_,
111         string calldata symbol_
112     ) external onlyRole(CONFIGURER_ROLE) whenNotPaused nonReentrant returns (address token) {
113         bytes memory initData = abi.encodeCall(
114             RWAToken.initialize,
115             (
116                 name_,
117                 symbol_,
118                 admin,
119                 vault,
120                 operator,
121                 configurer,
122                 address(tokenManager),
123                 ISanctionsList(sanctionsList),
124                 ICorporateActionOracle(actionOracle)
125             )
126         );
127
128         BeaconProxy proxy = new BeaconProxy(address(defaultBeacon), initData);
129         token = address(proxy);
130
131         tokenManager.setTokenRegistrationStatus(token, true);
132         emit TokenCreated(token, name_, symbol_, address(tokenManager));
133     }
```

Listing 2.29: src/RWATokenFactory.sol

2.3.2 Attestation issuance responsibility

Introduced by Version 1

Description The project states that the KYC process for users is outsourced to DEXs, and the project team only performs KYC on the DEX. Under this model, authorized attestations are still signed and distributed by the project. Therefore, the project should ensure that attestations are not issued to users who have not completed KYC on the DEX.

Feedback from the project The project team stated that KYC attestations will be strictly issued by an off-chain service and not given to non-KYC users.

2.3.3 Price parity assumption for `settleToken`

Introduced by Version 1

Description The contract `RWATokenManager` calculates the required settlement amount from the attested `USD` value using only the decimals of the settlement token. This logic assumes that each accepted settlement token maintains a value of 1 `USD`. If the settlement token used in the attestation materially depegs, users could potentially mint or redeem RWA exposure at an exchange rate that deviates from the intended market value.

Feedback from the project The project team stated that the settlement token will be pegged to 1 USD.

2.3.4 Divergent compliance paths across transfer flows

Introduced by Version 1

Description The current design applies stricter compliance checks in the function `mintWithSignature()` and the function `redeemWithSignature()` of the contract `RWATokenManager` than in the functions `transfer()` and `transferFrom()` of the contract `RWAToken`.

In the primary market flow, the manager verifies that `_msgSender()` is registered in the contract `IDRegistry`. In the secondary market flow, the token checks only the sanctions status and frozen status. Under this design, an address that cannot mint or redeem directly may still receive tokens through a peer-to-peer transfer. This model is acceptable if the intended compliance boundary applies only to primary issuance and redemption. Meanwhile, if the intended model requires only registered users to hold the token, the transfer path should align with the primary market path.

2.3.5 Ensure enough funds to pay users in `redeemVault`

Introduced by Version 1

Description In the contract `RWATokenManager`, the function `redeemWithSignature()` relies on the address `redeemVault` to provide the necessary settlement tokens for user redemptions. Therefore, the project should ensure that the address `redeemVault` has sufficient token balances and allowances.

2.3.6 Dust balance due to precision loss

Introduced by Version 1

Description In the contract RWATokenRebaseUpgradeable, the function balanceOf() calculates the user balance by dividing the raw balance by the rebase index. This division can lead to precision loss, and small residual amounts may remain as dust in the variable _rawBalances. These residual amounts cannot be displayed or utilized by the user since the division results in zero.

```
86    function balanceOf(
87        address account
88    ) public view virtual override returns (uint256) {
89        RebaseStorage storage $ = _getRebaseStorage();
90        return $._rawBalances[account] / $._rebaseIndex;
91    }
```

Listing 2.30: src/RWATokenRebaseUpgradeable.sol

2.3.7 Potential centralization risks

Introduced by Version 1

Description In this project, several privileged roles (e.g., DEFAULT_ADMIN_ROLE, OPERATOR_ROLE, CONFIGURER_ROLE, INVENTORY_ROLE, BURN_ROLE, and SIGNER_ROLE) can conduct sensitive operations, which introduces potential centralization risks. For example, the DEFAULT_ADMIN_ROLE can unpause the protocol after a freeze, replace critical addresses such as the vault, minter, sanctions list, and oracle, perform emergency withdrawals of any ERC-20 or ETH held by the token contracts, and grant or revoke any role, effectively giving it unrestricted control over the entire system. The OPERATOR_ROLE can unilaterally pause all minting and redemption activity globally or on a per-token basis, and freeze individual user accounts, blocking their transfers. The CONFIGURER_ROLE can delist RWA tokens (halting all transfers and suppressing supply/balance), trigger rebase operations that adjust all holder balances, configure rate limits, and deploy new RWA tokens through the factory. The INVENTORY_ROLE and BURN_ROLE can directly mint or burn tokens on behalf of the protocol without user-initiated signatures, while the SIGNER_ROLE controls the attestation flow that gates every user mint and redemption. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

2.3.8 Off-chain calculation for settleAmount

Introduced by Version 1

Description The project team is responsible for ensuring the correctness of settleAmount, which is computed off-chain before the transaction. On-chain, this value is verified against an estimated value stableCoinAmount to detect discrepancies. Deviations in either direction carry distinct consequences. An understated settleAmount may cause the transaction to revert, while an overstated value may result in direct financial losses for the user, as any excess amount paid is not refunded..

Feedback from the project The project stated that the correctness of `settleAmount` is ensured by the off-chain pricing service.

2.3.9 Asset backing assumption for inventory management functions

Introduced by `Version 1`

Description The functions `inventoryMint()` and `inventoryBurn()` allow authorized roles to mint or burn RWA tokens without requiring a corresponding settlement token transfer. This design implies that the total supply of the RWA token is synchronized with the off-chain underlying assets through external operational processes. The protocol remains secure under the assumption that the holders of the `INVENTORY_ROLE` and the `BURN_ROLE` ensure sufficient off-chain collateral exists to back the on-chain supply.

```
146   // Inventory Management
147   function inventoryMint(
148       address token,
149       uint256 amount,
150       address recipient
151   ) external nonReentrant whenNotPaused whenMintingNotPaused(token) onlyRole(INVENTORY_ROLE) {
152       if (amount == 0) revert ZeroAmount();
153       if (rwaTokenAccepted[token] == false) revert RWATokenNotRegistered();
154       if (exchangeSafeAccepted[recipient] == false) revert ExchangeSafeNotRegistered();
155
156       bytes32 userId = idRegistry.getRegisteredID(GM_IDENTIFIER, recipient);
157       if (userId == bytes32(0)) revert UserNotRegistered();
158
159       IRWAToken(token).mint(recipient, amount);
160       emit InventoryMint(recipient, token, amount);
161   }
162
163   function inventoryBurn(
164       address token,
165       uint256 amount
166   ) external nonReentrant whenNotPaused whenRedeemNotPaused(token) onlyRole(BURN_ROLE) {
167       if (amount == 0) revert ZeroAmount();
168       if (rwaTokenAccepted[token] == false) revert RWATokenNotRegistered();
169
170       IERC20(token).safeTransferFrom(_msgSender(), address(this), amount);
171       IRWAToken(token).burn(amount);
172       emit InventoryBurn(_msgSender(), token, amount);
173   }
```

Listing 2.31: src/RWATokenManager.sol

Feedback from the project The project team manages `INVENTORY_ROLE` and `BURN_ROLE`, both controlled by MPC. These two roles are permitted to mint and destroy RWA assets on demand, with settlement processed off-chain. During the minting process, tokens are expected to be transferred to the `BURN_ROLE` via standard transfers.

2.3.10 Off-chain handling after delisting

Introduced by Version 1

Description Once an RWA token is delisted, functions `balanceOf()` and `totalSupply()` return zero. As a result, holders can no longer transfer or redeem through the on-chain flow. If shareholders need to exit after delisting (e.g, buybacks or other settlement procedures), the project should handle the relevant processes off-chain.

Feedback from the project The project stated that migration and other delisting-related matters are handled off-chain.

2.3.11 Fixed decimal assumption in price normalization

Introduced by Version 1

Description The contract `SanityCheckOracle` assumes all Pyth price feeds share the same decimal exponent, using the fixed global variable `pythPriceDecimals` (i.e., 5) for normalization.

Feedback from the project The protocol only supports self-developed RWA stocks, and both the Pyth Pro feed and the protocol's own oracle consistently use 5 as the decimal value. Dynamically fetching per-feed decimals would increase gas costs and off-chain complexity, so a fixed value is used.

2.3.12 Allowance usage during rebasing process

Introduced by Version 1

Description In the contract `RWATokenRebaseUpgradeable`, allowances are stored in actual amount units after applying the rebase index and are not adjusted when the rebase index is updated. During a reverse split (i.e., `from > to`), balances decrease while existing allowances remain unchanged. Therefore, users should pay additional attention to existing allowances during reverse splits, since the approved amount may no longer reflect their intended post-rebase exposure.

```
131   function allowance(
132       address owner,
133       address spender
134   ) public view virtual override returns (uint256) {
135       RebaseStorage storage $ = _getRebaseStorage();
136       return $._allowances[owner][spender];
137   }
```

Listing 2.32: src/RWATokenRebaseUpgradeable.sol

```
189   function _transfer(
190       address from,
191       address to,
192       uint256 amount
193   ) internal virtual {
194       if (from == address(0)) revert ZeroAddress();
195       if (to == address(0)) revert ZeroAddress();
```

```
196
197         uint256 bal = balanceOf(from);
198         if (bal < amount) revert InsufficientBalance(bal, amount);
199
200         RebaseStorage storage $ = _getRebaseStorage();
201         uint256 rawAmount = amount * $._rebaseIndex;
202         $._rawBalances[from] -= rawAmount;
203         $._rawBalances[to] += rawAmount;
204
205         emit Transfer(from, to, amount);
206     }
```

Listing 2.33: src/RWATokenRebaseUpgradeable.sol

Feedback from the project The project stated that keeping allowances unchanged during a rebase is an intentional design choice. Since users' intent behind a given allowance cannot be determined on-chain, the team chose not to adjust allowances during rebases. This approach is consistent with Ondo.

2.3.13 Front-run risks of attestation submission

Introduced by `Version 2`

Description Version 2 introduces a notable change in business logic. The `user` field has been removed from the attestation, and the submission transaction no longer requires the `user` to be `tx.origin`. Therefore, a potential front-running risk exists when submitting attestations on-chain. If a legitimate attestation signed for a valid user is broadcast through a public mempool, a malicious actor could observe the pending transaction, submit the same attestation payload with a higher gas fee, and have it mined first. This would allow the malicious actor to use the legitimate user's valid attestation to obtain their RWA tokens.

Feedback from the project The project team will require partners to route transactions from valid users through privacy transaction channels.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS