BLOCKSEC

# Security Audit

# Report for RWADividendDistributor Contract

**Date:** May 8, 2026 **Version:** 1.0

**Contact:** contact@blocksec.com

# Contents


**Chapter 1 Introduction** **1**

1.1 About the Audit Target . . . 1
1.2 Disclaimer . . . 2
1.3 Procedure of Auditing . . . 2
1.3.1 Security Issues . . . 2
1.3.2 Additional Recommendation . . . 3
1.4 Security Model . . . 3

**Chapter 2 Findings** **5**

2.1 Security Issue . . . 5
2.1.1 Lack of check on cancelled actions . . . 5
2.2 Recommendation . . . 7
2.2.1 Add claim slippage protection . . . 7
2.2.2 Remove redundant code . . . 8
2.2.3 Strengthen payload length check . . . 8
2.2.4 Move nonce invalidation before verifications . . . 9
2.2.5 Restrict `DIVIDEND_DISTRIBUTOR_ROLE` to `ActionType.Dividend` . . . 9
2.2.6 Add a check on the claimable amount . . . 10
2.2.7 Adopt accurate variable name . . . 11
2.3 Note . . . 12
2.3.1 Potential centralization risks . . . 12
2.3.2 Global effect of expiration duration parameter changes . . . 12
2.3.3 Assumption of RWA payout tokens . . . 13
2.3.4 Off-chain calculation of cumulativeUnits . . . 13
2.3.5 Off-chain Merkle tree construction consistency . . . 14

## Report Manifest

| Item | Description |
| --- | --- |
| Client | Bitget |
| Target | RWADividendDistributor Contract |

## Version History

| Version | Date | Description |
| --- | --- | --- |
| 1.0 | May 8, 2026 | First release |

## Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-05-08 07:42:19+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 RWADividendDistributor Contract of Bitget.

The RWA EVM project introduces `RWADividendDistributor`, a new contract enabling trustless dividend distribution for tokenized real-world stock dividends. Dividends are distributed via a Merkle tree model. An operator submits a corporate action through contract `CorporateActionOracle`, which validates and anchors a new Merkle root on-chain. Users then claim entitlements by submitting a Merkle proof alongside an EIP-712 signature from a `SIGNER_ROLE` holder. Claims are cumulative, that each user's position in the tree represents total historical entitlement, and only the delta over prior claimed amounts is transferred. Two payout token types are supported: RWA tokens and standard ERC-20 stablecoins. Replay protection is enforced through a tree nonce and a bitmap-based unordered nonce. Per-channel pause controls allow operators to halt claims for individual (`rwaToken`, `payoutToken`) pairs independently.

Note this audit only focuses on the smart contracts in the following directories/files. Code prior to and including the baseline version 0, where applicable, is outside the scope of this audit and assumes to be reliable and secure.

- src/CorporateActionOracle.sol
- src/RWADividendDistributor.sol
- src/interfaces/ICorporateActionOracle.sol
- src/interfaces/IRWADividendDistributor.sol

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.

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.

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

| Project | Version | Commit Hash |
|---|---|---|
| rwa-evm | Version 0 | 0d09252e034e883dc67ea540705076360fdafbc4 |
| | Version 1 | adb8c4c456dfe2088e32e835273f95d79eb31085 |
| | Version 2 | 95ec4e045b38b24ea61cac9de0927a68834f1ba1 |

## 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.

**Table 1.1:** Vulnerability Severity Classification

| Impact \ Likelihood | *High* | *Low* |
|---|---|---|
| *High* | High | Medium |
| *Low* | Medium | Low |

Accordingly, the severity measured in this report are classified into three categories: **High**,

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

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

**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.
- **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 **one** potential security issue. Besides, we have **seven** recommendations and **five** notes.

- Low Risk: 1
- Recommendation: 7
- Note: 5

| ID | Severity | Description | Category | Status |
|---|---|---|---|---|
| 1 | Low | Lack of check on cancelled actions | Security Issue | Fixed |
| 2 | - | Add claim slippage protection | Recommendation | Confirmed |
| 3 | - | Remove redundant code | Recommendation | Fixed |
| 4 | - | Strengthen payload length check | Recommendation | Fixed |
| 5 | - | Move nonce invalidation before verifications | Recommendation | Fixed |
| 6 | - | Restrict `DIVIDEND_DISTRIBUTOR_ROLE` to `ActionType.Dividend` | Recommendation | Fixed |
| 7 | - | Add a check on the claimable amount | Recommendation | Fixed |
| 8 | - | Adopt accurate variable name | Recommendation | Fixed |
| 9 | - | Potential centralization risks | Note | - |
| 10 | - | Global effect of expiration duration parameter changes | Note | - |
| 11 | - | Assumption of RWA payout tokens | Note | - |
| 12 | - | Off-chain calculation of cumulativeUnits | Note | - |
| 13 | - | Off-chain Merkle tree construction consistency | Note | - |

The details are provided in the following sections.

## 2.1 Security Issue

### 2.1.1 Lack of check on cancelled actions

**Severity** Low

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In contract `CorporateActionOracle`, the function `cancelAction()` sets an action's status to `Status.Invalid`, indicating it should no longer be processed. However, function `updateAction()` allows any `CONFIGURER_ROLE` holder to submit the same action ID after cancellation, overwriting the stored action with new or unchanged parameters.

```
73    function cancelAction(
74        bytes32 id
```

```
75  ) external whenNotPaused onlyRole(CONFIGURER_ROLE) {
76      if (!_exists(id)) revert ActionNotFound(id);
77
78      Action storage action = actions[id];
79
80      if (action.base.status == Status.Executed) revert ActionAlreadyExecuted();
81      if (action.base.status == Status.Invalid) revert ActionInvalid();
82
83      action.base.status = Status.Invalid;
84
85      emit ActionCanceled(id);
86  }
```

**Listing 2.1:** src/CorporateActionOracle.sol

```
43  function updateAction(
44      bytes32 id,
45      Action calldata action
46  ) external whenNotPaused onlyRole(CONFIGURER_ROLE) {
47      _validateBaseInput(id, action);
48      _validatePayload(action.base.actionType, action.payload);
49
50      bool isNew = !_exists(id);
51
52      if (!isNew) {
53          Action storage existingAction = actions[id];
54          if (existingAction.base.status == Status.Executed) revert ActionAlreadyExecuted();
55      }
56
57      _updateAction(id, action);
58      exists[id] = true;
59
60      emit ActionUpdated(
61          id,
62          action.base.token,
63          action.base.ticker,
64          action.base.date,
65          action.base.category,
66          action.base.report,
67          action.base.actionType,
68          action.base.status,
69          action.payload
70      );
71  }
```

**Listing 2.2:** src/CorporateActionOracle.sol

**Impact** A cancelled action can be reactivated.

**Suggestion** Add a check to reject cancelled actions.

## 2.2 Recommendation

### 2.2.1 Add claim slippage protection

**Status** Confirmed

**Introduced by** Version 1

**Description** In the contract `RWADividendDistributor`, the function `_executeClaim()` handles two payout paths depending on the token type. When the payout token is an RWA token, the claimable amount (i.e., `claimableAmount`) is calculated as `claimableUnits / rebaseIndex`. Since the variable `rebaseIndex` can change due to rebase events, the actual amount received may be lower than expected, with no on-chain protection against the shortfall. It is recommended to add the parameter `minAllowedClaimAmount` to the function `claim()` to provide slippage protection.

```
289   function _executeClaim(
290       address rwaToken,
291       address payoutToken,
292       address account,
293       uint256 cumulativeUnits,
294       uint256 maxAllowedClaimAmount
295   ) internal {
296       uint256 previouslyClaimed = cumulativeClaimedUnits[rwaToken][payoutToken][account];
297       if (previouslyClaimed >= cumulativeUnits) revert NothingToClaim();
298
299       uint256 claimableUnits = cumulativeUnits - previouslyClaimed;
300
301       bool isRWAPayoutToken = allowedRWAPayoutTokens[rwaToken][payoutToken];
302       bool isStablePayoutToken = allowedStablePayoutTokens[rwaToken][payoutToken];
303
304       if (!isRWAPayoutToken && !isStablePayoutToken) revert PayoutTokenNotConfigured(payoutToken)
              ;
305       if (isRWAPayoutToken && isStablePayoutToken) {
306           revert PayoutTokenConfiguredInBothLists(payoutToken);
307       }
308
309       uint256 claimableAmount;
310       if (isRWAPayoutToken) {
311           // If payout token is an accepted RWA token, convert units to transfer amount using the
                  current rebase index.
312           uint256 rebaseIndex = RWAToken(payoutToken).rebaseIndex();
313           if (rebaseIndex == 0) revert InvalidPayoutToken(payoutToken);
314           claimableAmount = claimableUnits / rebaseIndex;
315       } else {
316           claimableAmount = claimableUnits;
317       }
318
319       if (claimableAmount > maxAllowedClaimAmount) revert ClaimAmountExceeded();
320
321       cumulativeClaimedUnits[rwaToken][payoutToken][account] = cumulativeUnits;
322       claimedTreeNonce[rwaToken][payoutToken][account] = merkleRootNonce[rwaToken][payoutToken];
323
```

```
324         IERC20(payoutToken).safeTransfer(account, claimableAmount);
325
326         emit DividendClaimed(rwaToken, payoutToken, account, claimableAmount);
327     }
```

**Listing 2.3:** src/RWADividendDistributor.sol

**Suggestion** Add a parameter and check against the actual claimable amount.

**Feedback from the project** The project confirmed this recommendation and stated that trading for the affected tokens will be paused before a rebase. Any change in the claimable amount caused by an updated `rebaseIndex` is intended behavior.

### 2.2.2 Remove redundant code

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the project, several unused errors and events are declared but never referenced. Specifically, the error `AddressUnchanged` and the event `DividendDistributorUpdated` are defined without serving any active purpose in the current implementation. It is recommended to remove them to improve code readability.

```
49    error AddressUnchanged(address current);
```

**Listing 2.4:** src/interfaces/ICorporateActionOracle.sol

```
53    event DividendDistributorUpdated(address indexed oldDividendDistributor, address indexed
          newDividendDistributor);
```

**Listing 2.5:** src/interfaces/ICorporateActionOracle.sol

**Suggestion** Remove the unused error and the event `DividendDistributorUpdated`.

### 2.2.3 Strengthen payload length check

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In contract `RWADividendDistributor`, function `validateCorporateActionAndSetMerkleRoot()` checks that the variable `payload` is non-empty and before decoding it as `(address, address, uint256)`, which requires exactly 96 bytes. A payload of an incorrect length passes this check but causes a revert with a generic decoding error (line 172), providing no actionable failure reason. It is recommended to check the `payload.length` equals the expected byte length to provide a clearer revert reason.

```
162   function validateCorporateActionAndSetMerkleRoot(
163       bytes32 id,
164       bytes32 newRoot,
165       bytes calldata payload
166   ) external whenNotPaused nonReentrant onlyRole(OPERATOR_ROLE) {
167       if (id == bytes32(0)) revert InvalidActionId();
```

```
168         if (newRoot == bytes32(0)) revert InvalidMerkleRoot();
169         if (payload.length == 0) revert EmptyPayload();
170
171         // payload = (rwaToken, payoutToken, dividendPerShare), dividendPerShare not used.
172         (address rwaToken, address payoutToken,) = abi.decode(payload, (address, address, uint256))
                ;
```

**Listing 2.6:** src/RWADividendDistributor.sol

**Suggestion** Replace the check `payload.length == 0` with `payload.length != 96` and update the revert error accordingly.

### 2.2.4 Move nonce invalidation before verifications

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the contract `RWADividendDistributor`, the function `claim()` invokes the function `_useUnorderedNonce()` after Merkle proof validation and signature verification. Given that Merkle proof verification and signature verification consume a significant amount of gas, it is recommended to move the call to `_useUnorderedNonce()` to an earlier position.

```
212         _validateClaimAgainstCurrentTree(
213             rwaToken, payoutToken, account, cumulativeUnits, request.merkleRoot, merkleProof
214         );
215
216         _verifyClaimSignature(request, signature);
217
218         _useUnorderedNonce(account, request.signatureNonce);
```

**Listing 2.7:** src/RWADividendDistributor.sol

**Suggestion** Move the invocation of the function `_useUnorderedNonce()` before Merkle proof validation and signature verification.

### 2.2.5 Restrict `DIVIDEND_DISTRIBUTOR_ROLE` to `ActionType.Dividend`

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the contract `CorporateActionOracle`, the function `validateAndExecuteAction()` allows the `DIVIDEND_DISTRIBUTOR_ROLE` to execute actions with arbitrary types. The contract `RWADividendDistributor` exclusively uses `ActionType.Dividend`, but this restriction is not enforced in the contract `CorporateActionOracle`. It is recommended to add an on-chain check in the function `validateAndExecuteAction()` that enforces `ActionType.Dividend` for all callers holding `DIVIDEND_DISTRIBUTOR_ROLE`.

```
88    function validateAndExecuteAction(
89        bytes32 id,
90        bytes32 dataHash
91    ) external whenNotPaused {
```

```
92        if (!_exists(id)) revert ActionNotFound(id);
93
94        Action storage action = actions[id];
95
96        if (msg.sender != action.base.token && !hasRole(DIVIDEND_DISTRIBUTOR_ROLE, msg.sender)) {
97            revert NotActionCaller();
98        }
99
100       if (action.base.status == Status.Executed) revert ActionAlreadyExecuted();
101       if (action.base.status != Status.Valid) revert ActionInvalid();
102
103       if (!_isWithinExecutionWindow(action.base.date)) revert ActionNotInExecutionWindow();
104
105       bytes32 expectedHash = _computeDataHash(action.base.actionType, action.payload);
106       if (dataHash != expectedHash) revert InvalidDataHash();
107
108       action.base.status = Status.Executed;
109
110       emit ActionExecuted(
111           id,
112           action.base.token,
113           action.base.ticker,
114           action.base.date,
115           action.base.category,
116           action.base.report,
117           action.base.actionType,
118           action.base.status,
119           action.payload
120       );
121   }
```

**Listing 2.8:** src/CorporateActionOracle.sol

**Suggestion** Add a check that restricts callers holding `DIVIDEND_DISTRIBUTOR_ROLE` to `ActionType.Dividend` only.

### 2.2.6 Add a check on the claimable amount

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the contract `RWADividendDistributor`, when dividends are paid in RWA tokens, the claimable amount is calculated using the `rebaseIndex` at the time of claim. A reverse split may therefore cause the resulting `claimableAmount` to round down to zero. In that case, the function permits a zero-value claim to succeed and advances the user's cumulative claim state even though no meaningful payout is received.

```
310       if (isRWAPayoutToken) {
311           // If payout token is an accepted RWA token, convert units to transfer amount using the
                  current rebase index.
312           uint256 rebaseIndex = RWAToken(payoutToken).rebaseIndex();
313           if (rebaseIndex == 0) revert InvalidPayoutToken(payoutToken);
```

```
314             claimableAmount = claimableUnits / rebaseIndex;
315         } else {
```

**Listing 2.9:** src/RWADividendDistributor.sol

```
200     function claim(
201         ClaimRequest calldata request,
202         bytes32[] calldata merkleProof,
203         bytes calldata signature
204     ) external whenNotPaused nonReentrant {
205         _validateClaimRequest(request);
```

**Listing 2.10:** src/RWADividendDistributor.sol

**Suggestion** Add a check on the variable `claimableAmount`.

### 2.2.7 Adopt accurate variable name

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the contract `RWADividendDistributor`, the function `claim()` enforces a claim limit through the variable `maxAllowedClaimAmount`. However, this name is misleading. Each claim settles the full delta between the user's latest cumulative entitlement and previously claimed amount, rather than applying a generic upper bound.

```
289     function _executeClaim(
290         address rwaToken,
291         address payoutToken,
292         address account,
293         uint256 cumulativeUnits,
294         uint256 maxAllowedClaimAmount
295     ) internal {
296         uint256 previouslyClaimed = cumulativeClaimedUnits[rwaToken][payoutToken][account];
297         if (previouslyClaimed >= cumulativeUnits) revert NothingToClaim();
298
299         uint256 claimableUnits = cumulativeUnits - previouslyClaimed;
300
301         bool isRWAPayoutToken = allowedRWAPayoutTokens[rwaToken][payoutToken];
302         bool isStablePayoutToken = allowedStablePayoutTokens[rwaToken][payoutToken];
303
304         if (!isRWAPayoutToken && !isStablePayoutToken) revert PayoutTokenNotConfigured(payoutToken)
                ;
305         if (isRWAPayoutToken && isStablePayoutToken) {
306             revert PayoutTokenConfiguredInBothLists(payoutToken);
307         }
308
309         uint256 claimableAmount;
310         if (isRWAPayoutToken) {
311             // If payout token is an accepted RWA token, convert units to transfer amount using the
                    current rebase index.
312             uint256 rebaseIndex = RWAToken(payoutToken).rebaseIndex();
313             if (rebaseIndex == 0) revert InvalidPayoutToken(payoutToken);
```

```
314             claimableAmount = claimableUnits / rebaseIndex;
315         } else {
316             claimableAmount = claimableUnits;
317         }
318
319         if (claimableAmount > maxAllowedClaimAmount) revert ClaimAmountExceeded();
320
321         cumulativeClaimedUnits[rwaToken][payoutToken][account] = cumulativeUnits;
322         claimedTreeNonce[rwaToken][payoutToken][account] = merkleRootNonce[rwaToken][payoutToken];
323
324         IERC20(payoutToken).safeTransfer(account, claimableAmount);
325
326         emit DividendClaimed(rwaToken, payoutToken, account, claimableAmount);
327     }
```

**Listing 2.11:** src/RWADividendDistributor.sol

**Suggestion** Rename the variable to a name that reflects its actual semantics.

## 2.3 Note

### 2.3.1 Potential centralization risks

**Introduced by** Version 2

**Description** In this project, several privileged roles (e.g., `DEFAULT_ADMIN_ROLE`, `OPERATOR_ROLE`, `CONFIGURER_ROLE`, and `SIGNER_ROLE`) can conduct sensitive operations, which introduces potential centralization risks. For example, `OPERATOR_ROLE` can drain contract funds via functions `emergencyWithdraw()` and `emergencyWithdrawERC20Exact()` without delay. The `CONFIGURER_ROLE` can replace the `actionOracle` address or manipulate payout token allowlists, and `SIGNER_ROLE` holders can authorise arbitrary dividend claims. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

**Feedback from the project** The project will use MPC or multisig wallets for privileged roles to mitigate centralization risks.

### 2.3.2 Global effect of expiration duration parameter changes

**Introduced by** Version 1

**Description** In the contract `CorporateActionOracle`, the function `setExpirationDuration()` modifies the state variables `beforeExpirationDuration` and `afterExpirationDuration`, so changes take effect on all existing valid actions immediately. Specifically, reducing the variable `afterExpirationDuration` may render a previously executable action unexecutable. Conversely, expanding the variable `beforeExpirationDuration` may allow an action to be executed earlier than originally intended. Since action dates are stored without their window parameters, the original execution window cannot be recovered after a change. The project should invoke the function carefully and be aware of its impact.

```
197         uint256 actionDate
```

```
198   ) internal view returns (bool) {
199       uint256 nowTs = block.timestamp;
200       uint256 start = 0;
201       if (actionDate > beforeExpirationDuration) {
202           start = uint256(actionDate) - uint256(beforeExpirationDuration);
203       }
204       uint256 end = uint256(actionDate) + uint256(afterExpirationDuration);
205       return nowTs >= start && nowTs < end;
206   }
```

**Listing 2.12:** src/CorporateActionOracle.sol

**Feedback from the project** The project acknowledged that updates to these state variables have a global effect on existing valid actions. As these parameters exist solely to reconcile timing differences between the off-chain system and the on-chain execution window, changes are expected to be rare.

### 2.3.3 Assumption of RWA payout tokens

**Introduced by** Version 1

**Description** The contract `RWADividendDistributor` accepts any token that exposes the function `rebaseIndex()` as an RWA payout token. The project should ensure that any such token behaves consistently with the contract `RWAToken`. Specifically, the token is expected to adopt an economic model where the actual balance equals the raw balance divided by the variable `rebaseIndex`. Additionally, the token is expected to revert when the function `transfer()` is called with an amount of zero.

**Feedback from the project** The project will explicitly document this requirement in the operation documentation, stating that any integration involving RWA payout tokens must ensure token behavior is consistent with the expected RWA token model.

### 2.3.4 Off-chain calculation of cumulativeUnits

**Introduced by** Version 1

**Description** In the contract `RWADividendDistributor`, the actual meaning of `cumulativeUnits` depends on the payout token type. For stable payout tokens, `cumulativeUnits` represents the final transfer amount denominated in the payout token's own decimals. For RWA payout tokens, `cumulativeUnits` represents a raw internal unit amount, and the actual transfer amount is derived on-chain at claim time by dividing by the current `rebaseIndex`. The contract does not perform decimal normalization or validate the off-chain scaling logic. Therefore, the project should ensure that `cumulativeUnits` is calculated consistently for each payout token type and should account for possible truncation dust caused by integer division in the RWA payout path.

**Feedback from the project** The off-chain system snapshots user balances at a specific block and calculates each user's payout amount based on the recorded balance and the dividend rate. For stable payout tokens, the payout amount is written directly into the Merkle leaf as `cumulativeUnits`. For RWA payout tokens, the payout amount is converted into internal share units using the `rebaseIndex` at snapshot time and stored as `cumulativeUnits`.

### 2.3.5 Off-chain Merkle tree construction consistency

**Introduced by** Version 1

**Description** The project should ensure that the off-chain Merkle tree builder uses the same eligibility, accounting, and hashing conventions as the on-chain verifier in the contract `RWADividendDistributor`. Specifically, the builder should clearly define which accounts are included in each dividend distribution and preserve previously accrued but unclaimed entitlement when rebuilding the cumulative tree for a new round. Additionally, the proof construction must match the on-chain verification logic exactly. The leaf encoding must match `keccak256(abi.encodePacked(rwaToken, payoutToken, account, cumulativeUnits))`, the pair hashing must be commutative (smaller hash placed first), and the odd-node carry-up convention must be consistent. Any deviation from these conventions may prevent valid beneficiaries from claiming dividends.

```
329   function _verifyMerkleProof(
330       bytes32[] calldata proof,
331       bytes32 root,
332       bytes32 leaf
333   ) private pure returns (bool valid) {
334       assembly {
335           let ptr := proof.offset
336           for { let end := add(ptr, mul(0x20, proof.length)) } lt(ptr, end) { ptr := add(ptr, 0
                  x20) } {
337               let node := calldataload(ptr)
338
339               switch lt(leaf, node)
340               case 1 {
341                   mstore(0x00, leaf)
342                   mstore(0x20, node)
343               }
344               default {
345                   mstore(0x00, node)
346                   mstore(0x20, leaf)
347               }
348
349               leaf := keccak256(0x00, 0x40)
350           }
351
352           valid := eq(root, leaf)
353       }
354   }
```

**Listing 2.13:** src/RWADividendDistributor.sol

**Feedback from the project** Users holding the specified RWA token at the snapshot block are eligible for inclusion in the Merkle tree as a leaf node, provided their calculated dividend amount meets the required minimum threshold. Only holders captured in the snapshot are considered.

Accounts that are frozen, blacklisted, placed on sanctions lists, or otherwise restricted due to compliance controls will be excluded from the Merkle tree based on off-chain eligibility determination rules.

Any unclaimed balances from previous distributions are carried forward into the latest

Merkle tree, allowing users to claim all outstanding historical dividends in a single transaction.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS