






## Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Bridge	Documentation quality	Medium 
Timeline	2024-01-15 through 2024-01-23	Test quality	Medium 
Language	Solidity	Total Findings	24 Fixed: 17 Acknowledged: 6 Mitigated: 1
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review	High severity findings ⓘ	2 Fixed: 2
Specification	<a href="#">Chromia Docs</a> 	Medium severity findings ⓘ	5 Fixed: 3 Acknowledged: 2
Source Code	<ul style="list-style-type: none"> <li><a href="https://gitlab.com/chromaway/postchain-eif">https://gitlab.com/chromaway/postchain-eif</a> </li> <li><a href="#">#f73af2f</a> </li> </ul>	Low severity findings ⓘ	10 Fixed: 8 Acknowledged: 1 Mitigated: 1
Auditors	<ul style="list-style-type: none"> <li>Ibrahim Abouzied Auditing Engineer</li> <li>Hytham Farah Auditing Engineer</li> <li>Andy Lin Senior Auditing Engineer</li> </ul>	Undetermined severity findings ⓘ	1 Acknowledged: 1
		Informational findings ⓘ	6 Fixed: 4 Acknowledged: 2

## Summary of Findings

The Chromia Token Bridge allows users to bridge tokens back and forth between EVM-compatible chains and Chromia. The bridge follows the Lock & Mint model for L1 to L2 transfers, and the Burn & Release model for L2 to L1 transfers. L1 to L2 transfers are handled by Chromia listening to token deposit events, and L2 to L1 transfers are performed by submitting a Merkle Proof to L1 verifying that the tokens have been burned on L2.

Among other features, token withdrawal requests undergo a holding period, allowing the contract owner to flag and deny any suspicious transactions before withdrawal execution. The contract also has a separate withdrawal mechanism in the event of a mass exit: a scenario in which all normal bridging activity is halted, and users are only permitted to withdraw their token balances per a specified Chromia node snapshot.

Among the notable issues discovered in the audit, some token withdrawals will not be possible for transferable tokens (CHRM-1). The current implementation for mass exits allows for duplicate withdrawals (CHRM-3). Validator management faces issues as well. Former validators can forge transactions for old block heights (CHRM-2), anyone can forge a transaction if all validators are removed for a given block height (CHRM-4), and a new validator can forge blocks by front-running updates to the list of validators (CHRM-5).

The Chromia team was in communication and helped answer all of our questions throughout the audit. The documentation was helpful, though the codebase could benefit from more in-line comments and NatSpec. Regarding the test suite, we were unable to find tests for withdrawals during mass exits. It is important that all core features are tested before deployment. Additionally, we would like to emphasize that this audit only covers the Ethereum/EVM-Compatible chain side of the Token Bridge. The Chromia side of the bridge is outside the scope of the audit.

**Update:** The Chromia team has fixed, acknowledged, or mitigated all findings.

ID	DESCRIPTION	SEVERITY	STATUS
CHRM-1	Token Owner Unable to Withdraw From the Postchain	• High ⓘ	Fixed
CHRM-2	Invalid Withdrawal by Former Validators	• High ⓘ	Fixed
CHRM-3	Potential Double Spend upon Mass Exit	• Medium ⓘ	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
CHRM-4	Contract Can Be Drained if There Are No Validators	• Medium ⓘ	Fixed
CHRM-5	Validator Addition & Removal Is Vulnerable to Front-Running	• Medium ⓘ	Fixed
CHRM-6	Risk of Running Out of Gas on Old Withdrawals	• Medium ⓘ	Fixed
CHRM-7	Privileged Roles and Ownership	• Medium ⓘ	Acknowledged
CHRM-8	Risk of Updating Balance with Failed Transfer in <code>TokenBridge.fund()</code>	• Low ⓘ	Fixed
CHRM-9	Signature Malleability	• Low ⓘ	Acknowledged
CHRM-10	<code>_balances</code> Inconsistency	• Low ⓘ	Fixed
CHRM-11	Merkle Proof Library Implementation Weaknesses	• Low ⓘ	Fixed
CHRM-12	Risk of Gas Insufficiency for Users with Excessive Tokens During Mass Exit	• Low ⓘ	Mitigated
CHRM-13	User Can Create Misleading Event	• Low ⓘ	Fixed
CHRM-14	Missing Input Validation	• Low ⓘ	Fixed
CHRM-15	Ownership Can Be Renounced	• Low ⓘ	Fixed
CHRM-16	Critical Role Transfer Not Following Two-Step Pattern	• Low ⓘ	Fixed
CHRM-17	Missing Initializers	• Low ⓘ	Fixed
CHRM-18	Tokens May Return Empty Metadata	• Informational ⓘ	Acknowledged
CHRM-19	Short Dispute Period	• Informational ⓘ	Fixed
CHRM-20	Expensive Gas Usage with Nested Loop	• Informational ⓘ	Fixed
CHRM-21	Application Monitoring Can Be Improved by Emitting More Events	• Informational ⓘ	Fixed
CHRM-22	Unlocked Pragma	• Informational ⓘ	Fixed
CHRM-23	Upgradability	• Informational ⓘ	Acknowledged
CHRM-24	Re-Orgs May Lead to Double Spend	• Undetermined ⓘ	Acknowledged

## Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

### **i** Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

### Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence

- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

## Methodology

1. Code review that includes the following
  1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
  2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
  3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
  1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
  2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

## Scope

The scope of the audit only covers the EVM-Compatible chain side of the Token Bridge. The Chromia side of the bridge is out of scope.

### Files Included

- `contracts/utils/cryptography/*`
- `contracts/Data.sol`
- `contracts/Postchain.sol`
- `contracts/TokenBridge.sol`
- `contracts/Validator.sol`

### Files Excluded

- `contracts/token/*`
- `contracts/utils/test/*`
- `contracts/NFTBridge.sol`

## Findings

### CHRM-1 Token Owner Unable to Withdraw From the Postchain

• High ⓘ

Fixed



#### Update

Marked as "Fixed" by the client. Addressed in: `d85d98c4bfa9faa464d3c6ef2f824457008027a2`. The client provided the following explanation:

We removed the specific `_ALICElimits` out of the smart contract

**File(s) affected:** `TokenBridge.sol`

**Description:** The `TokenBridge.withdraw()` function verifies the `_ALICElimits` before allowing a withdrawal. The `_ALICElimits` tracks a user's deposited amount on the bridge. However, if a user bridges tokens to the Postchain and subsequently transfers those tokens, the token recipient on the Postchain will be unable to withdraw because their `_ALICElimits` might be zero if they have never deposited tokens. In other words, the owners of the bridged token are at risk of being unable to redeem their token back in the original chain.

**Exploit Scenario:** Alice deposits 100 ERC-20 tokens into the Postchain and then transfers these bridged tokens to Bob. Bob's attempt to withdraw fails since his `_ALICElimits` is zero.

**Recommendation:** In discussions with the team during the audit, they indicated plans to remove this feature. If it is indeed removed as unnecessary, this issue will be considered resolved.

## CHRM-2 Invalid Withdrawal by Former Validators

• High ⓘ

Fixed

### ✓ Update

Marked as "Fixed" by the client. Addressed in: `18934185980ccd30e418f3c8a3daab25364566f1`. The client provided the following explanation:

"Agree, this is know issue and we're developing a new version of token bridge that will using chromia anchoring on evm chain they we can validate chromia block header like your suggestion in next version. It will take time then we will not fix this in this version. For the time being, we will only allow the latest validator list and do migration for old withdrawal on chromia side as below steps (like Migration.sol):

1. transfer the ownership of the validator contract to the smart contract
2. write a method to migration all the pending withdrawal on Chromia side: update validator list to old one, call `withdrawrequest` for each pending withdrawal, update the validator list to current one.
3. transfer the ownership of the validator contract back to admin/owner (the smart contract should have a method to transfer the ownership of the validator contract) Notes: We aim to get rid of the manual process described in the future. Instead we will solve it by letting the client ask the new validators for a signature of the old block. Client can then use this to construct a new witness to pass to the contract.

**File(s) affected:** `TokenBridge.sol`

**Description:** The `TokenBridge` bridge contract depends on the validation of signatures from validators of a certain height to confirm that a `blockRid` is legitimate, ensuring the withdrawal event is part of the Postchain. However, there is a concern that former validators from previous heights could collaborate to withdraw all funds by endorsing an invalid `blockRid`. Although the detailed implementation is beyond the scope of this audit, as per the [whitepaper](#), providers are required to stake Chromia tokens as collateral, which is forfeited if nodes under a provider's control exhibit misconduct. This implies a risk that former validators, who no longer have stakes in the Chromia collateral, might extract all the funds from the bridge without facing penalties.

**Exploit Scenario:** Consider the following example:

1. At height `h1`, Alice and Bob are the only validators.
2. At height `h2`, the validators change to Carol and David, with Alice and Bob withdrawing their stakes.
3. Alice and Bob collude to sign a malicious `blockRid` and generate an invalid withdrawal request.
4. Following the `WITHDRAW_OFFSET` period, Alice and Bob can extract all the funds from the bridge.

**Recommendation:** A potential solution involves re-designing the contract and its off-chain components to anchor the Postchain's block IDs to the contract. Consequently, the bridge would be able to verify the legitimacy of block IDs based on their anchoring, rather than relying on on-demand signatures.

## CHRM-3 Potential Double Spend upon Mass Exit

• Medium ⓘ

Acknowledged

### ⓘ Update

Marked as "Acknowledged" by the client.

The team added `block.number` instead of `height` to the `WithdrawRequest` event in the commit `6761d52`. It might be hard to filter out all pending withdrawals if only based on `block.number` as there could be withdraw request before triggering the mass exit. Addressed in: `e1c352cee28c5d3ecd9c9bad8f528cff21c54ea4`.

We also not that the validation in `withdraw()` is still missing. If a user submits a withdrawal request and then a mass exit triggers, the user will still be able to call `withdraw()`.

The client provided the following explanation:

When setting mass-exit the admin operator need to preparing carefully and gain agreement with all the current chormia validators. After that, the chromia blockchain will be halt and be maintained as read-only data. Basically, the mass-exit will be only triggered only and only one time, the update function is just for update the mass-exit if admin operator set the wrong mass-exit information. And because the account snapshot merkle tree will be updated whenever there's change on the account balance then the old withdraw before mass exit block is still valid.

1. We already checked on the `withdrawRequest()`.
2. `massExitBlock` is the chromia block height and `WITHDRAW_OFFSET` period is on ethereum block so it's not appropriate to do that validation.
3. As I mentioned in the note, the update mass-exit block is just for admin to update if wrong setting only. Still not find a difference way to handle the wrong setting mass-exit block (some user might already withdraw before updating).
4. I added more block height into the event."

**File(s) affected:** `TokenBridge.sol`

**Description:** A mass exit can be triggered when there is suspicious activity on the Chromia chain. All further withdrawal requests will have to be done on a snapshot block assembled periodically by Chromia nodes. Users will no longer be able to submit withdrawal requests for blocks after the snapshot block.

However, the current implementation is at risk of enabling double withdrawals. This could happen if a mass exit is assigned a height that precedes some already processed withdrawal requests. In such cases, a user might be able to withdraw funds twice: once through the standard withdrawal process and again using the `withdrawBySnapshot()` function to withdraw from the snapshot. This can also happen by calling `withdrawBySnapshot()` twice if `updateMassExitBlock()` is called, as the new `_snapshots[stateProof.leaf]` will be `false`.

The `TokenBridge.pendingWithdraw()` function could potentially mitigate this risk. However, it would require the `WITHDRAW_OFFSET` to be lengthy enough for the admin to intervene. Based on discussions with the team, they appear to be planning to update the `WITHDRAW_OFFSET` to delay withdrawals by approximately one week on Ethereum, which should give the admin ample time to halt withdrawals at higher heights.

**Recommendation:** We believe there is no direct way to prevent this scenario; therefore, we would like the team to clearly outline their strategy for managing such situations. We also recommend the following mitigations:

1. In `withdraw()`, validate that the withdrawal did not occur past the mass exit, e.g. `require(!isMassExit || wd.height < massExitBlock.height)`.
2. Do not allow `triggerMassExit()` to assign a `massExitBlock` to a block further in the past than the `WITHDRAW_OFFSET` period. `WITHDRAW_OFFSET` should always allow time to mark all withdrawals that occurred after the mass exit to be parked as `Status.Pending`.
3. Either avoid updating the `massExitBlock`, or find a different way for marking withdrawals as consumed.
4. Incorporate the `height` data into the `WithdrawRequest` event, which would facilitate faster recognition and sorting of the pertinent withdrawal requests, ensuring the efficient identification of withdrawals that require calls to `TokenBridge.pendingWithdraw()`.

## CHRM-4 Contract Can Be Drained if There Are No Validators

• Medium ⓘ Fixed

### ✓ Update

Marked as "Fixed" by the client. Addressed in: `f08c9b2b2d585477114375264cb0a92f524c407b`.

**File(s) affected:** `Validator.sol`

**Description:** The `Validator.removeValidator()` function does not verify whether it is removing the last validator at a specific height. Consequently, when the final validator of a given height is removed, the `validatorHeights` array still includes that height, even though no validators remain. This creates a vulnerability where users can submit false data to create a fake `blockRid`, which will still pass the `validator.isValidSignatures()` check, as no signatures would be required in this scenario.

**Recommendation:**

1. Either require for there to always be at least one validator for a given block height, or delete the block height from the `validatorHeights[]` array once the last validator is removed.
2. Do not allow `isValidSignatures()` to return true if there are no signers. Consider enforcing a minimum number of signers.

## CHRM-5

### Validator Addition & Removal Is Vulnerable to Front-Running

• Medium ⓘ Fixed

### ✓ Update

Marked as "Fixed" by the client. Addressed in: `18934185980ccd30e418f3c8a3daab25364566f1`. The client provided the following explanation:

Along with [CHRM-2](#), we simplify chromia validator list management and only update the validator list in a unique contract method to avoid front-running. Note: This front-running vuln is good finding. Thank you for pointing out this. For the current version, these operations be managed by admin/owner account then we need to update the validator list, admin/owner account wil temporarily pause the TokenBridge smart contract first and unpause it after they complete validator list update. Chromia also need to clear communication about the step-by-step operations admin need to do when updating the validator list. And in the next version of extened token bridge the validator list be managed by postchain/chromia system and we need to consider about these kine of front-running vulns.

**File(s) affected:** `Validator.sol`

**Description:** `addValidator()` and `removeValidator()` manage the array of approved validators for a given block height. A transaction can be approved before the full list of validators is up to date by front-running or reordering transactions. For example, if only one validator has signed a block, and it is the only one that has been added to `validators[height]` (with the remaining `addValidator()` calls waiting in the mempool), it is possible for a user to sandwich their call such that the sole validator is considered a majority.

Additionally, if a `removeValidator()` call at height `n` followed by an `addValidator()` call at height `n+1` enter the mempool, they can be reordered such that the `addValidator()` call is processed first, making the `removeValidator()` call impossible to complete for the previous height.

**Exploit Scenario:**

1. Alice becomes a validator for the newest block.
2. The contract owner submits `addValidator()` calls to resubmit the full list of validators into the mempool.
3. Alice reorders the transactions to sandwich a forged withdrawal right after her `addValidator()` call.
4. Given that she is the only validator listed for her withdrawal transaction, the withdrawal goes through and drains the contract.

#### Recommendation:

1. Create a `setValidators()` function such that all of the approved validators can be added and all unapproved validators can be removed for a given height simultaneously.
2. Avoid submitting calls to `removeValidator()` and `addValidator()` for different heights to the mempool simultaneously. Alternatively, create a function that can perform these operations for different heights.

## CHRM-6 Risk of Running Out of Gas on Old Withdrawals

• Medium ⓘ Fixed

### ✓ Update

Marked as "Fixed" by the client. Addressed in: `18934185980ccd30e418f3c8a3daab25364566f1`. The client provided the following explanation:

Due to the change in the way we manage the validator list on [CHRM-2](#) then this issue should not happen anymore.

**File(s) affected:** `TokenBridge.sol`, `Validator.sol`

**Description:** The function `Validator._getValidatorHeight()` searches from the most recent height down to 0 to find the closest height that is less than the target height. This is done to get the right set of validators for a specific height. However, there is a risk that this search could use up all the gas available for very old heights, especially in extreme cases.

The team mentioned that how often the validators change depends on their staking and is not known yet. If we assume a block time of 10 seconds and changes in the validator set every 32 blocks (like Ethereum), this means a change happens every 320 seconds. This results in about  $60 * 60 * 24 * 365 / 320 = 98,550$  changes each year. If checking `validatorHeights[i]` costs at least 200 gas (only the SLOAD gas cost, in practice it will be more), then finding a height from a year ago would need at least  $98,550 * 200 = 19,710,000$  gas. With Ethereum's current average gas limit of 30M per block, the gas could run out after about 1.5 years.

The risk is marked as medium, not high, because it's likely that validator set changes happen less often in practice. Ethereum changes its validator set every 32 blocks for randomness, but Chromia might not do this as frequently, though their exact change rate isn't known. Also, users usually withdraw their funds soon rather than waiting a long time.

**Recommendation:** To fix this, consider using a binary search method instead of looking through each index one by one.

## CHRM-7 Privileged Roles and Ownership

• Medium ⓘ Acknowledged

### i Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

Not code commit need, just need clear communication with the community

**File(s) affected:** `TokenBridge.sol`

**Description:** Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.

- `TokenBridge.setBlockchainRid()`: A setter for `blockchainRid`.
- `TokenBridge.pause()` and `TokenBridge.unpause()`: Pauses and unpauses the contract. **A compromised owner can block all withdrawals by permanently pausing the contract.**
- `TokenBridge.allowToken()`: Add a new supported ERC-20 token.
- `TokenBridge.increaseALICELimit()`: Manually increase a user's `ALICELimit`.
- `TokenBridge.triggerMassExit()`, `TokenBridge.postponeMassExit()`, and `TokenBridge.updateMassExitBlock()`: Manage the triggering of a mass exit, in which users can redeem their tokens at a given snapshot.
- `TokenBridge.pendingWithdraw()` and `TokenBridge.unpendingWithdraw()`: Disable a withdrawal request for review during a dispute period. **A compromised owner could deny a user from withdrawing their tokens.**
- `TokenBridge.fund()`: Adds more tokens to the contract to facilitate withdrawals.
- `TokenBridge.emergencyWithdraw()`: Allows the withdrawal of any amount of tokens after a specified timestamp. **A compromised owner can drain the wallet once the `emergencyTimestamp` has passed.**
- `Validator.addValidator()` and `Validator.removeValidator()`: Manages the list of validators for a given block height. **A compromised owner can modify the list of validators to deny withdrawals or forge their own withdrawals.**

**Recommendation:** This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

## CHRM-8

### Risk of Updating Balance with Failed Transfer in `TokenBridge.fund()`

• Low ⓘ Fixed

#### ✓ Update

Marked as "Fixed" by the client. Addressed in: `b9fdc108000120373366309843f8432fe109f0c0`.

**File(s) affected:** `TokenBridge.sol`

**Description:** The `TokenBridge.fund()` function uses the `transferFrom()` method without checking the return value, contrary to the EIP-20 standard stating:

Callers MUST handle false from returns (bool success). Callers MUST NOT assume that false is never returned!

If the call was not completed successfully, the `_balances` mapping would be incorrectly updated.

**Recommendation:** Use `safeTransferFrom()` instead of `transferFrom()`, or, ensure the return value of the `transferFrom()` function is checked.

## CHRM-9 Signature Malleability

• Low ⓘ Acknowledged

#### ⓘ Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

Some hardware wallet will encounter the issue so we comment the check I think

**File(s) affected:** `ECDSA.sol`

**Description:** Signature malleability is a characteristic of the ECDSA (Elliptic Curve Digital Signature Algorithm) cryptographic algorithm used in Ethereum and Solidity. It refers to the ability to produce different valid signatures for the same message by modifying the signature parameters. This can occur due to certain mathematical properties of the ECDSA algorithm.

In the `ECDSA` contract, the line `require(uint256(s) <= 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0, "ECDSA: invalid signature 's' value");` has been commented out. This introduces the risk of signature malleability. From the perspective of this audit, the impact appears limited because using a malleable signature still implies that the validator has signed. We did not identify any obvious exploits in the contracts stemming from this vulnerability. However, this omission makes the `ECDSA` library potentially hazardous for reuse.

**Recommendation:** Uncomment the line to reinstate the validation against the `s` value in signatures. For more details, you can also refer to the guidelines provided in Open Zeppelin reference implementation [here](#).

## CHRM-10 `_balances` Inconsistency

• Low ⓘ Fixed

#### ✓ Update

Marked as "Fixed" by the client. Addressed in: `5d57d1faa0ae7af1959d4f11493278272359a3a3`.

**File(s) affected:** `TokenBridge.sol`

**Description:** The `TokenBridge` contract maintains a record of the total locked balance of a token using the `_balances` mapping. This value increases with `deposit()` and `fund()` calls and decreases with `withdraw()` calls. However, the `withdrawBySnapshot()` and `emergencyWithdraw()` functions transfer tokens out without reducing the `_balances` value. These functions seem to be designed for emergencies, so the necessity of updating `_balances` may depend on the recovery steps that are expected to follow. If the use of these functions indicates that the bridge is no longer functional and users are expected to migrate to new contracts, the current approach might be acceptable.

Moreover, even if `_balances` is not updated, transferring tokens that are not locked in the contract remains impossible. Therefore, the impact of the `_balances` inconsistency is considered low. However, there could be instances where the public `_balances` variable is used by off-chain components, which falls outside the scope of this audit. The team should investigate any potential issues that may arise from this.

**Recommendation:** The team should clearly define the recovery steps to be taken after a mass exit or emergency withdrawal. This clarification will help determine whether it is necessary to adjust the `_balances` values in such scenarios.

## CHRM-11 Merkle Proof Library Implementation Weaknesses

• Low ⓘ Fixed

#### ✓ Update

Marked as "Fixed" by the client. Addressed in: 84339ae2a113fe5a52ea636189f394f193892602 . The client provided the following explanation:

1. Fixed
2. Fixed
3. **Second Pre-image Attack:** Actually, the prefix on chromia gtv merkle tree is virtual tree that adding prefix into the value, not prefix of the Merkle tree hash proof. So the prefix is just added to calculate the leaf's hash properly in smart contract in `Hash.hashGtvBytes32Leaf()`, `Hash.hashGtvBytes64Leaf` and `Hash.hashGtvIntegerLeaf()`. So there's no fix for this."

**File(s) affected:** `MerkleProof.sol`, `Hash.sol`

**Description:** We observed several potential weaknesses in the current `MerkleProof` library implementation:

1. **Invalid position Data:** The `position` input of the `verify()` and `verifySHA256()` functions can be manipulated when `position` exceeds the number of Merkle tree leaves. For example, in a Merkle tree with only four leaves, providing a position value of 3, 7, or 11 would have an identical impact.
2. **Empty Leaf Bytes:** Leaves with empty bytes can pass the inclusion check in the `verify()` function. While this may be valid from a sparse Merkle tree's perspective (as empty leaves are virtually part of the tree), it can lead to unexpected validation successes.
3. **Second Pre-image Attack** (see: [Merkle tree wiki](#)): Instead of validating a leaf, one could provide an intermediate node pretending to be the leaf data, thereby conducting a second preimage attack.

We did not find any exploits leveraging these weaknesses during the audit:

1. **Invalid position Data:** Although invalid position data can be used, the Merkle proof validation only passes if the leaf node is indeed included. The contracts that call the `MerkleProof` do not rely further on the position data.
2. **Empty Leaf Bytes:** We consider it impossible to find a pre-image of empty bytes that can be decoded and effectively used.
3. **Second Pre-image Attack:** We consider it impossible to find a pre-image of the intermediate nodes that can be decoded and effectively used. Also, from the discussion with the team, the nodes of the GTV merkle tree are with prefix but just the prefix is not part of the verification within the contract code.

Despite no exploits being found that leverage these weaknesses, the team should be aware and consider fixing them to prevent future changes from introducing issues.

**Recommendation:** We suggest adding some protections:

1. **Invalid position Data:** Verify that the `position` is within the range `[0, 2^proofs.length - 1]`.
2. **Empty Leaf Bytes:** Consider verifying that the leaf cannot be empty bytes.
3. **Second Pre-image Attack:** Consider adding prefixes verification to the intermediate and leaf nodes as part of the merkle proof verification function.

## CHRM-12

### Risk of Gas Insufficiency for Users with Excessive Tokens During Mass Exit

• Low ⓘ

Mitigated

#### Update

Marked as "Acknowledged" by the client. Addressed in: 8d4443e379c363b2dad46511ed76aec5194d1a64 . The client provided the following explanation:

Yes, like you said in practical the number of tokens a user can hold is likely much lower than 3000. If this happen, we will inform user to send several tokens into another account to reduce the total number of token on each account to avoid gas limit. Further, we think that we need to check that the snapshot token must be the allowed token in the smart contract as well. Otherwise, the withdrawal might be error.

**File(s) affected:** `TokenBridge.sol`

**Description:** The `TokenBridge.withdrawBySnapshot()` function uses a for-loop to return all tokens belonging to a user. If a user possesses too many tokens as per the snapshot data, there's a risk of gas insufficiency to complete the withdrawal of all their tokens.

For each token transfer, there is a basic cost involving two `SSTORE` operations to update the balances of both the sender and the recipient. The `SSTORE` typically costs 2100 for accessing cold storage, plus 2900 for value updates (see: [doc](#)). Therefore, the minimum cost would be  $5000 * 2 = 10,000$  gas per token. Considering Ethereum's average block gas limit of 30M, a user can't hold more than 3000 tokens. However, this calculation overlooks other gas costs, so in reality, the practical limit for the number of tokens a user can hold is likely much lower.

**Recommendation:** The team should consider conducting simulations to determine the actual token limit and inform users accordingly. This will help ensure that users do not store an excessive number of tokens in a single account, avoiding potential withdrawal issues.

## CHRM-13 User Can Create Misleading Event

• Low ⓘ

Fixed



### ✓ Update

Marked as "Fixed" by the client. Addressed in: `f2ff3e6d88b4799656cd36ffe921ab5d893e1778` .

**File(s) affected:** `TokenBridge.sol`

**Description:** In the `TokenBridge.deposit()` function a user is free to supply the `ft3_account_id` variable to anything they would like since it is only used in the emitted event `DespositedERC20()` . A malicious user may therefore pass an incorrect value in order to emit a misleading event.

**Recommendation:** Either validate the `ft3_account_id` variable, or retrieve it from other information such as `msg.sender` .

## CHRM-14 Missing Input Validation

• Low ⓘ Fixed

### ✓ Update

Marked as "Fixed" by the client. Addressed in: `0495fba7f28bbd5d348a80b2da9a4083931471a6` .

**File(s) affected:** `Hash.sol` , `TokenBridge.sol` , `Validator.sol`

**Related Issue(s):** [SWC-123](#)

**Description:** Input validation is crucial for maintaining system security and reliability. It helps to prevent errors and potential attacks by restricting the range of inputs that attackers can use. We recommend adding input validations in the following areas:

- `Hash.hashGtvBytes64Leaf()` : This function does not verify that `value.length == 64` , which is necessary to ensure it hashes only 64-byte inputs. Currently, callers can provide inputs with more than 64 bytes.
- `TokenBridge.allowToken()` : The `token` parameter can be a zero address. Allowing a zero address as a permitted token could unexpectedly make verification bypasses easier.
- `TokenBridge.pendingWithdraw()` and `unpendingWithdraw()` : The `_hash` input can be an empty `bytes32` . This could lead to the `_withdraw[]` mapping having a value for the empty key. Generally, having values for empty key mappings is risky, as they can be used accidentally in unforeseen ways.
- `TokenBridge.emergencyWithdraw()` : Consider checking that both the `beneficiary` and `token` are not zero addresses. The risk of mistakenly assigning the wrong `beneficiary` is particularly high, as it could result in transfers to an uncontrolled destination.
- `TokenBridge.initialize()` : Validate that `_validator` is a non-zero address.
- `TokenBridge.postponeMassExit()` : Set the previous `massExitBlock` value to `(0, 0)` .
- `TokenBridge._withdrawRequest()` : Validate that `blockchainRid` has been initialized.
- `Validator.addValidator()` : Consider validating that the `_validator` is not an empty address to prevent the accidental addition of a zero address as a validator.

**Recommendation:** Implement the suggested validations as detailed in the description above.

## CHRM-15 Ownership Can Be Renounced

• Low ⓘ Fixed

### ✓ Update

Marked as "Fixed" by the client. Addressed in: `2e299a0226bfd9492f2e8406ddeeb5ca365d0be3` .

**File(s) affected:** `TokenBridge.sol` , `Validator.sol`

**Description:** If the owner renounces their ownership, all ownable contracts will be left without an owner. Consequently, any function guarded by the `onlyOwner` modifier will no longer be able to be executed.

**Recommendation:** Confirm that this is the intended behavior. If not, override and disable the `renounceOwnership()` function in the affected contracts. For extra security, consider using a two-step process when transferring the ownership of the contract (e.g. `Ownable2Step` from OpenZeppelin).

## CHRM-16 Critical Role Transfer Not Following Two-Step Pattern

• Low ⓘ Fixed

### ✓ Update

Marked as "Fixed" by the client. Addressed in: `4e95c3b8f43d2e549b77b8a3b80486801c7823d2` .

**File(s) affected:** `TokenBridge.sol` , `Validator.sol`

**Description:** The owner of the contracts can call `transferOwnership()` to transfer the ownership to a new address. If an uncontrollable address is accidentally provided as the new owner address then the contract will no longer have an active owner, and functions with the `onlyOwner` modifier can no longer be executed.

**Recommendation:** Consider using OpenZeppelin's `Ownable2Step` contract to adopt a two-step ownership pattern in which the new owner must accept their position before the transfer is complete.

## CHRM-17 Missing Initializers

• Low ⓘ Fixed

### ✓ Update

Marked as "Fixed" by the client. Addressed in: `f165a56b8a5fc18fabeb595166f117cd522013a3`.

**File(s) affected:** `TokenBridge.sol`

**Description:** The `TokenBridge` contract is missing a call to the initializers of `PausableUpgradeable` and `ReentrancyGuardUpgradeable`.

**Recommendation:** Call `__Pausable_init()` and `__ReentrancyGuard_init()` in the initializer of the `TokenBridge` contract.

## CHRM-18 Tokens May Return Empty Metadata

• Informational ⓘ Acknowledged

### i Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

The metadata token information is just additional information for the token. In the logic handle on chromia side only depends on the token address so the token metadata is not mandatory.

**File(s) affected:** `TokenBridge.sol`

**Description:** Token metadata, such as the name, symbol, and decimal count, are emitted in some events. The metadata is gathered in calls to `_getTokenInfo()`. However, the function silently fails for tokens that do not support metadata. This results in event emissions with empty values for token metadata.

**Recommendation:** If it is a requirement for tokens to support metadata, validate that tokens support the required functions in `allowToken()`.

## CHRM-19 Short Dispute Period

• Informational ⓘ Fixed

### ✓ Update

Marked as "Fixed" by the client. `withdrawOffset` can be made into an immutable variable. Addressed in: `78bd673aebcc285c64559b273fa7efa7ece7485c`.

**File(s) affected:** `TokenBridge.sol`

**Description:** `WITHDRAW_OFFSET` specifies the number of blocks that a user must wait between the approval of their withdrawal request and executing the withdrawal. This dispute period allows the Chromia team to address any suspicious activity. `WITHDRAW_OFFSET` is currently initialized to `2` blocks, which makes sense for testing the contract, but is inadequate for deploying to production.

**Recommendation:** Increase `WITHDRAW_OFFSET` to a dispute period sufficient for addressing any suspicious activity before deploying to production. Consider making it an immutable value so that it is assignable on deployment, removing the need to modify the contract for test and production environments.

## CHRM-20 Expensive Gas Usage with Nested Loop

• Informational ⓘ Fixed

### ✓ Update

Marked as "Fixed" by the client. Addressed in: `627d0c65f0a14e768dc933f2c1069af5560e813e`.

**File(s) affected:** `TokenBridge.sol`, `Validator.sol`

**Description:** The `Validator.isValidSignatures()` function uses a nested loop to check the validity of signatures. However, a single-layer loop could be enough if the client code ensures that the addresses in the `signers` input are in the correct order.

Currently, the code looks like this:

```
for (uint i = 0; i < signatures.length; i++) {
    for (uint k = 0; k < signers.length; k++) {
```

```

require(isValidator(height, signers[k]), "Validator: signer is not validator");
if (!_isValidSignature(hash, signatures[i], signers[k])) {
    _actualSignature++;
    require(signers[k] > _lastSigner, "Validator: duplicate signature or signers is out of order");
    _lastSigner = signers[k];
    break;
}
}
}

```

It can be simplified to something like this:

```

for (uint i = 0; i < signatures.length; i++) {
    require(isValidator(height, signers[i]), "Validator: signer is not validator");
    if (!_isValidSignature(hash, signatures[i], signers[i])) {
        _actualSignature++;
        require(signers[i] > _lastSigner, "Validator: duplicate signature or signers is out of order");
        _lastSigner = signers[i];
    }
}

```

This change requires the `signers` input to be in the same order as the addresses that signed the signatures. By removing the nested loop, this approach could help save gas.

Meanwhile, if the system can unify the format of the message to be signed, it would streamline the process. Currently, the contract allows both `prefixedProof` and `hash` as valid message formats for signing. However, if only one format is used, it could simplify the contract's operations. With a unified format, the need for the `signers` array could be eliminated, and the code could simply rely on the signatures to recover the signer's address, without having to cross-check against `signers[i]`. At present, because there are two potential addresses that can be recovered (one from `prefixedProof` and one from `hash`), the code requires an additional reference for verification. Removing this need would also eliminate the necessity for referencing `signers`.

**Recommendation:** Consider removing the nested loop if the client code can be updated accordingly. Additionally, if the message format can be standardized, think about eliminating the `signers` input from the function too.

## CHRM-21

### Application Monitoring Can Be Improved by Emitting More Events

• Informational ⓘ Fixed

#### ✓ Update

Marked as "Fixed" by the client. Addressed in: `e1c352cee28c5d3ecd9c9bad8f528cff21c54ea4`.

**File(s) affected:** `TokenBridge.sol`

**Description:** In order to validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transitions can be logged, which is beneficial for monitoring the contract, and also tracking eventual bugs or hacks. Below we present a non-exhaustive list of functions that could emit events to improve application management:

- `TokenBridge.initialize()`
- `TokenBridge.setBlockchainRid()`
- `TokenBridge.allowToken()`
- `TokenBridge.increaseALICELimit()`
- `TokenBridge.triggerMassExit()`
- `TokenBridge.postponeMassExit()`
- `TokenBridge.updateMassExitBlock()`
- `TokenBridge.pendingWithdraw()`
- `TokenBridge.unpendingWithdraw()`
- `TokenBridge.withdrawRequest()`

**Recommendation:** Consider emitting the events.

## CHRM-22 Unlocked Pragma

• Informational ⓘ Fixed

#### ✓ Update

Marked as "Fixed" by the client. Addressed in: `09f66721fafafdb09a56bab1cb3da323092ae851`.

**File(s) affected:** `All files`

**Description:** Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. The caret ( `^` ) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

**Recommendation:** For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto a specific Solidity version.

## CHRM-23 Upgradability

• Informational ⓘ Acknowledged

### **i** Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

Not code commit need, just need clear communication with the community

**File(s) affected:** `TokenBridge.sol`

**Description:** While upgradability is not a vulnerability in itself, users should be aware that the `TokenBridge` can be upgraded at any time. This audit does not guarantee the behavior of future contract upgrades.

**Recommendation:** The fact that the contract can be upgraded and reasons for future upgrades should be communicated to users beforehand.

## CHRM-24 Re-Orgs May Lead to Double Spend

• Undetermined ⓘ Acknowledged

### **i** Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

Not code commit need, just need clear communication with the community

**File(s) affected:** `TokenBridge.sol`, `Validator.sol`

**Description:** A reorg is when a sequence of blocks that were part of the blockchain gets substituted with a different sequence of blocks at some later point in time. An event that was emitted before a reorg may no longer be included in the chain after the reorg. Therefore, a malicious attacker could initiate a deposit, wait for it to be confirmed by the Chromia validators, and then reorg the blockchain such that the deposit is no longer included. This will allow them to claim funds on the destination chain without ever having deposited funds in the source chain.

The Ethereum blockchain is only considered finalized after 2 epochs (roughly 12 minutes). Before then, while it is extremely difficult to execute a reorg it is not impossible. Hence, the team should be aware of this possibility and be able to mitigate against it.

Though the functioning of the chromia validators is outside the scope of the audit, the consequences of the attack are severe and hence included in the report.

**Recommendation:** Ensure that the state of the source chain is finalized before accepting an event on the destination chain.

## Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

# Appendix

## File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

## Contracts

- db2...831 ./contracts/Validator.sol
- e24...e0a ./contracts/Postchain.sol
- 390...230 ./contracts/Data.sol
- d53...568 ./contracts/TokenBridge.sol
- 007...e0f ./contracts/utils/cryptography/Hash.sol
- cb3...bf9 ./contracts/utils/cryptography/ECDSA.sol
- 4aa...05f ./contracts/utils/cryptography/MerkleProof.sol

## Tests

- 49c...8fd ./test/erc20.test.ts
- 07f...4c8 ./test/utils.ts
- 4d5...29d ./test/bridge.test.ts
- 1e4...919 ./test/utility.test.ts
- 00f...544 ./test/nft.test.ts

# Toolset

The notes below outline the setup and steps performed in the process of this audit.

## Setup

Tool Setup:

- [Slither](#) [v0.8.3](#)

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

# Automated Analysis

## Slither

Slither analyzed 43 contracts with 102 detectors, finding 180 results. The relevant findings have been included in the main section of the report.

# Test Suite Results

The test data was gathered by running `yarn test`.

### Token Bridge Test

#### Validators

- ✓ Admin can update validator(s) successfully (168ms)

#### Deposit

- ✓ User can deposit ERC20 token to target smartcontract (179ms)

#### Emergency Withdraw

- ✓ Emergency Withdraw (236ms)

#### Withdraw by normal user

- ✓ User can request withdraw by providing properly proof data (1555ms)

#### Withdraw via smart contract

- ✓ Integrate with smart contract (1410ms)

#### Mass Exit

- ✓ only admin can manage mass exit (116ms)

### Token

#### Mint

- ✓ Should mint some tokens

#### Transfer

- ✓ Should transfer tokens between users
- ✓ Should fail to transfer with low balance

#### Non Fungible Token

##### Deposit NFT

- ✓ User can deposit NFT to target smartcontract (181ms)

##### Withdraw NFT

- ✓ User can withdraw NFT by providing properly proof data (993ms)

#### Utility Test

##### Utility

###### hash

- ✓ Non-empty node sha3 hash function
- ✓ Right empty node sha3 hash function
- ✓ Left empty node sha3 hash function
- ✓ All empty node
- ✓ hash gtv integer leaf 0
- ✓ hash gtv integer leaf 1
- ✓ hash gtv integer leaf 127
- ✓ hash gtv integer leaf 128
- ✓ hash gtv integer leaf 168
- ✓ hash gtv integer leaf 255
- ✓ hash gtv integer leaf 256
- ✓ hash gtv integer leaf 1023
- ✓ hash gtv integer leaf 1024
- ✓ hash gtv integer leaf 32769
- ✓ hash gtv integer leaf 1234567890

###### Merkle Proof

- ✓ Verify valid merkle proof properly
- ✓ Invalid merkle proof

###### SHA256 Merkle Proof

- ✓ Verify valid SHA256 merkle proof properly
- ✓ Invalid SHA256 merkle proof due to incorrect merkle root

30 passing (10s)

## Code Coverage

The test coverage was gathered by running `yarn coverage`. The code coverage has room for improvement. There are no tests for `TokenBridge.withdrawBySnapshot()`, which is used to facilitate withdrawals during a mass exit. It is crucial that all core features are properly tested. We encourage the team to strive for 100% code coverage.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
<b>contracts/</b>	77.78	57.14	88	81.58	
Data.sol	100	100	100	100	
NFTBridge.sol	78.72	54	84.62	83.1	... 189,190,191
Postchain.sol	100	100	100	100	
TokenBridge.sol	64.94	51.72	82.61	72.81	... 304,308,309
Validator.sol	83.72	63.33	100	88.24	... 72,73,74,77
<b>contracts/token/</b>	46.15	25	55.56	46.15	
AliceToken.sol	0	0	0	0	14,15,16,20,21

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
ERC721Mock.sol	50	100	60	50	14,22
TestToken.sol	100	50	100	100	
<b>contracts/utis/cryptograp hy/</b>	85.11	79.17	88.89	87.72	
ECDSA.sol	90	62.5	100	90.91	22
Hash.sol	100	100	100	100	
MerkleProof.sol	68.42	75	66.67	76	... 51,52,53,55
<b>contracts/utis/test/</b>	100	100	100	100	
TestDelegator.sol	100	100	100	100	
TokenBridgeDelegator.sol	100	100	100	100	
All files	78.2	58.82	85.71	81.74	

## Changelog

- 2024-01-23 - Initial report
- 2024-02-20 - Final report

## About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over \$200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

### Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

### Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

### Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

#### **Disclaimer**

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.



# Quantstamp