CERTIK

KiKiFinance iBTC.w

Security Assessment

CertiK Assessed on Apr 29th, 2025

CertiK Assessed on Apr 29th, 2025

KiKiFinance iBTC.w

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	EVM Compatible	Formal Verification, Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 04/29/2025	N/A

CODEBASE

https://github.com/KikiFinance/ibtc.w-contract

View All in Codebase Page

COMMITS

- 3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a
- 21f4681636cb4dcf4fb7366380f251259128e480

View All in Codebase Page

Highlighted Centralization Risks

- Contract upgradeability
- Privileged role can remove users' tokens
- Transfers can be paused

Vulnerability Summary

17	9	0	8	0
Total Findings	Resolved	Partially Resolved	Acknowledged	Declined

3	Centralization	3 Acknowledged	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
0	Major		Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
3	Medium	2 Resolved, 1 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.

6	Minor	5 Resolved, 1 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
5	Informational	2 Resolved, 3 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | KIKIFINANCE IBTC.W

Summary

Executive Summary

Vulnerability Summary

Codebase

Audit Scope

Approach & Methods

Review Notes

Overview

External Dependencies

Addresses

Privileged Functions

Findings

KII-02 : Centralized Control of Contract Upgrade

KII-03 : Centralized Balance Manipulation

KII-04 : Centralized Related Risk

KII-05 : `initialize()` Is Unprotected

KII-14 : Lack of Validation for Bitcoin Deposit Identifiers and Unbounded Mint Amounts

KII-18 : Missing Recovery Mechanism for Burned Tokens

KII-07 : Status of Current Account in User Manager Not Checked When Minting Tokens

KII-08 : Missing Zero Address Validation

KII-09 : `addDstChains()` Does Not Validate Whether the Destination Chain Is `block.chainId`

KII-10 : Inactive Pause Functionality in Bridge and RequestManager Contracts

KII-15 : Confirmation on Hash Collision Risks and Bridge Design

KII-17 : Missing Input Validation

KII-01 : Confirmation on Off-Chain Handler

KII-12 : Missing Emit Events

KII-13 : Adding Extra Data in `confirmBurnRequest` Will Result in a Different Hash for the Request

KII-16 : Confirmation on the Status of Cross-Chain Request

KII-19 : Confirmation on `addMintRequest()`

Formal Verification

Considered Functions And Scope

Verification Results

Appendix

Disclaimer

CODEBASE | KIKIFINANCE IBTC.W

Repository

https://github.com/KikiFinance/ibtc.w-contract

Commit

- 3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a
- 21f4681636cb4dcf4fb7366380f251259128e480

AUDIT SCOPE | KIKIFINANCE IBTC.W

12 files audited ● 6 files with Acknowledged findings ● 6 files without findings

ID	Repo	File	SHA256 Checksum
● CMK	KikiFinance/ibtc.w-contract	contracts/ChainManager.sol	4e5f0d03659c35239ab69c59c77fc7ef7ff4965753a7b2764f4913f0ddec714c
● FCS	KikiFinance/ibtc.w-contract	contracts/FeeConfigStore.sol	b398b689bd7b12ceb3b57ef5fd8c6fe3e25227e0e6bbca568fdd6e7cb23cd733
● RMK	KikiFinance/ibtc.w-contract	contracts/RequestManager.sol	f93992a76d9bf99f11ca7e7d53b85f70826dc2403a4badfdd7bc73f0d61600a1
● UMK	KikiFinance/ibtc.w-contract	contracts/UserManager.sol	9e8c5a2b663807247135f2a813915a8e8e4c4f8b4339cc21e77027fc9d9ade9b
● BTC	KikiFinance/ibtc.w-contract	contracts/iBTCwBridge.sol	e3f753866a37110f3b86c990b47e1dade6786056fd67b3c7a69e9af8f2aedbe9
● BTT	KikiFinance/ibtc.w-contract	contracts/iBTCwToken.sol	ddf00a4ee74ad735f877ef45583fe2670bd8fa166676c20548f7b7b16e041081
● CKF	KikiFinance/ibtc.w-contract	contracts/ChainManager.sol	66490554232f9a7fdfe03f70fb0b702b32c7915a4dc3e3aa0561bdb03eb0d29a
● FCF	KikiFinance/ibtc.w-contract	contracts/FeeConfigStore.sol	96ad2c39ab58675071fcbfc89f52ab9f435df8b9c50418b0a80d33aaa0a79e77
● RKF	KikiFinance/ibtc.w-contract	contracts/RequestManager.sol	8d2230ff386b46ffcf02031eba97538a6ac316a60b7d0a89a3ce4b87b6862931
● UKF	KikiFinance/ibtc.w-contract	contracts/UserManager.sol	01df5b551be4177d9a562196de91e274d72b47424e1ceac95caee3c1348fba5d
● BTF	KikiFinance/ibtc.w-contract	contracts/iBTCwBridge.sol	7b3715e63b288a0bc4e149256bb6ebb51b519e302879573a813d354eba1200bc
● BCT	KikiFinance/ibtc.w-contract	contracts/iBTCwToken.sol	38f044a337708d3e6f8863f7e354a294997de13e1bd632ef0e099de7b5b2943e

APPROACH & METHODS | KIKIFINANCE IBTC.W

This report has been prepared for KiKiFinance to discover issues and vulnerabilities in the source code of the KiKiFinance iBTC.w project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis, Formal Verification, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTES | KIKIFINANCE IBTC.W

Overview

KiKiFinance - iBTC.w is designed for multi-chain use. It supports minting, burning, and cross-chain transfers (via burn and mint), enabling `ibtcw` tokens to move across different blockchains.

The files currently under review include：

- ChainManager.sol
- FeeConfigStore.sol
- iBTCwBridge.sol
- iBTCwToken.sol
- RequestManager.sol
- UserManager.sol

External Dependencies

In **KiKiFinance - iBTC.w**, the module inherits or uses a few of the depending injection contracts or addresses to fulfill the need of its business logic. The scope of the audit treats third party entities as black boxes and assume their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets.

Addresses

The following addresses interact at some point with specified contracts, making them an external dependency. All of the following values are initialized either at deployment time or by specific functions in smart contracts.

iBTCwBridge

- `ibtcw`, `minter`, `feeStore`, `userManager`, `requestManager`, `chainManager`, `feeRecipient`.

We assume these contracts or addresses are valid and non-vulnerable actors and implementing proper logic to collaborate with the current project.

Also, the following library/contract are considered as the third-party dependencies:

- @openzeppelin/contracts-upgradeable/
- @openzeppelin/contracts/

Privileged Functions

In the **KiKiFinance - iBTC.w** project, the privileged roles are adopted to ensure the dynamic runtime updates of the project,

which are specified in the `Centralization` finding.

The advantage of those privileged roles in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community. It is also worth noting the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the private keys of the privileged accounts are compromised, it could lead to devastating consequences for the project.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

FINDINGS | KIKIFINANCE IBTC.W

17	0	3	0	3	6	5
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for KiKiFinance iBTC.w. Through this audit, we have uncovered 17 issues ranging from different severity levels. Utilizing the techniques of Static Analysis, Formal Verification & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
KII-02	**Centralized Control Of Contract Upgrade**	**Centralization**	**Centralization**	**Acknowledged**
KII-03	**Centralized Balance Manipulation**	**Centralization**	**Centralization**	**Acknowledged**
KII-04	**Centralized Related Risk**	**Centralization**	**Centralization**	**Acknowledged**
KII-05	`initialize()` Is Unprotected	Logical Issue	Medium	Resolved
KII-14	Lack Of Validation For Bitcoin Deposit Identifiers And Unbounded Mint Amounts	Volatile Code	Medium	Acknowledged
KII-18	Missing Recovery Mechanism For Burned Tokens	Logical Issue	Medium	Resolved
KII-07	Status Of Current Account In User Manager Not Checked When Minting Tokens	Logical Issue	Minor	Resolved
KII-08	Missing Zero Address Validation	Volatile Code	Minor	Resolved
KII-09	`addDstChains()` Does Not Validate Whether The Destination Chain Is `block.chainId`	Coding Issue	Minor	Resolved
KII-10	Inactive Pause Functionality In Bridge And RequestManager Contracts	Design Issue	Minor	Resolved
KII-15	Confirmation On Hash Collision Risks And Bridge Design	Logical Issue	Minor	Acknowledged

ID	Title	Category	Severity	Status
KII-17	Missing Input Validation	Volatile Code	Minor	Resolved
KII-01	Confirmation On Off-Chain Handler	Design Issue	Informational	Resolved
KII-12	Missing Emit Events	Coding Style	Informational	Resolved
KII-13	Adding Extra Data In `confirmBurnRequest` Will Result In A Different Hash For The Request	Inconsistency	Informational	Acknowledged
KII-16	Confirmation On The Status Of Cross-Chain Request	Volatile Code	Informational	Acknowledged
KII-19	Confirmation On `addMintRequest()`	Design Issue	Informational	Acknowledged

KII-02 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	Centralization	contracts/ChainManager.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 9; contracts/FeeConfigStore.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 7; contracts/RequestManager.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 9; contracts/UserManager.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 9; contracts/iBTCwBridge.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 19; contracts/iBTCwToken.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 9	Acknowledged

Description

In these contracts, the role `DEFAULT_ADMIN_ROLE` has the authority to update the implementation contract behind the UUPS contract.

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow a hacker to take advantage of this authority and change the implementation contract which is pointed by proxy and therefore execute potential malicious functionality in the implementation contract.

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

Short Term:

A combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;

AND

- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

- Renounce the ownership and never claim back the privileged role;
  OR
- Remove the risky functionality.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[KiKiFinance, 04/23/2025]: The team acknowledged the issue and decided to remain unchanged in the scope of the audit.

[CertiK, 04/27/2025]: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

KII-03 | CENTRALIZED BALANCE MANIPULATION

Category	Severity	Location	Status
Centralization	Centralization	contracts/iBTCwToken.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 30, 35	Acknowledged

Description

In the contract `iBTCwToken` , the role `MINTER_ROLE` has the authority to update the token balance of an arbitrary account without sanity restriction.

Any compromise to the `MINTER_ROLE` account may allow a hacker to take advantage of this authority and manipulate users' balances.

Recommendation

We recommend the team makes efforts to restrict access to the private key of the privileged account. A strategy of multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to mint more tokens or engage in similar balance-related operations.
Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently *fully* resolve the risk:

Short Term:

A multi signature (⅔, ⅗) wallet *mitigate* the risk by avoiding a single point of key management failure.

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers' addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A DAO for controlling the operation *mitigate* the risk by applying transparency and decentralization.

- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the multi-signers' addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

The following actions can *fully* resolve the risk:

- Renounce the ownership and never claim back the privileged role.
  OR
- Remove the risky functionality.
  OR
- Add minting logic (such as a vesting schedule) to the contract instead of allowing the owner account to call the sensitive function directly.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[KiKiFinance, 04/23/2025]: The team acknowledged the issue and decided to remain unchanged in the scope of the audit.

[CertiK, 04/27/2025]: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

KII-04 | CENTRALIZED RELATED RISK

Category	Severity	Location	Status
Centralization	Centralization	contracts/ChainManager.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 25, 27, 37; contracts/FeeConfigStore.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 33, 35, 47; contracts/RequestManager.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 45, 47, 68, 81, 91; contracts/UserManager.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 34, 36, 48, 66, 74, 81; contracts/iBTCwBridge.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 68, 75, 199, 223, 238; contracts/iBTCwToken.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 28, 30, 35, 41, 45, 54, 58	Acknowledged

Description

In the contract `ChainManager`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and add or remove destination chains, as well as authorize contract upgrades for new implementation addresses.

In the contract `FeeConfigStore`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and authorize upgrades for a new implementation, set fee configuration, and set the fee recipient address.

In the contract `RequestManager`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and set the bridge address or authorize smart contract upgrades.

In the contract `RequestManager`, the role `bridge` has authority over the functions shown in the diagram below. Any compromise to the `bridge` account may allow the hacker to take advantage of this authority and add a new request to the system, confirm a pending request, or confirm a request with additional data.

In the contract `UserManager`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and unlock a qualified user, add a qualified user, edit qualified user's deposit and withdrawal addresses, remove a qualified user, lock a qualified user, and authorize smart contract upgrades.

In the contract `iBTCwToken`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and unpause the contract, add to blacklist, remove addresses from the blacklist, and authorize smart contract upgrades.

In the contract `iBTCwToken`, the role `MINTER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `MINTER_ROLE` account may allow the hacker to take advantage of this authority and mint tokens to a specified address or burn tokens from an address.

In the contract `iBTCwToken`, the role `PAUSER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `PAUSER_ROLE` account may allow the hacker to take advantage of this authority and pause the smart contract.

In the contract `iBTCwBridge`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and set the minter address or authorize contract upgrades with admin role.

In the contract `iBTCwBridge`, the role `minter` has authority over the functions shown in the diagram below. Any compromise to the `minter` account may allow the hacker to take advantage of this authority and confirm burn requests with transaction details, confirm and mint tokens for mint requests, and confirm crosschain requests.

The `ChainManager`, `FeeConfigStore`, `RequestManager`, `UserManager`, `iBTCwToken`, and `iBTCwBridge` contracts

inherit from `AccessControlUpgradeable`. The `DEFAULT_ADMIN_ROLE` has authority over the following functions:

- `grantRole`
- `revokeRole`

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.

  OR

- Remove the risky functionality.

Alleviation

[KiKiFinance, 04/23/2025]: The team acknowledged the issue and decided to remain unchanged in the scope of the audit.

[CertiK, 04/27/2025]: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

KII-05 | `initialize()` IS UNPROTECTED

Category	Severity	Location	Status
Logical Issue	Medium	contracts/ChainManager.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 9~53; contracts/FeeConfigStore.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 7~56; contracts/RequestManager.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 9; contracts/UserManager.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 9; contracts/iBTCwBridge.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 19~20; contracts/iBTCwToken.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 9~10	Resolved

Description

These logic contracts does not protect the initializer. An attacker can front-run the `initialize` call and assume ownership of the logic contract. Once in control, the attacker can perform privileged operations, misleading users into believing that they are interacting with the legitimate owner of the upgradeable contract.

Recommendation

We recommend adding

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() initializer {}
```

The addition will prevent the function `initialize()` from being called directly in the implementation contract, but the proxy will still be able to `initialize()` its storage variables.

Alleviation

[KiKiFinance, 04/25/2025]: The team heeded the advice and resolved the issue by adding `initializer` to the constructor in commit `21f4681636cb4dcf4fb7366380f251259128e480` .

KII-14 LACK OF VALIDATION FOR BITCOIN DEPOSIT IDENTIFIERS AND UNBOUNDED MINT AMOUNTS

Category	Severity	Location	Status
Volatile Code	Medium	contracts/iBTCwBridge.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 87~88	Acknowledged

Description

The `addMintRequest` function allows eligible users to submit a mint request. According to the inline comment, "The deposit txid and output index are recorded to prevent reusing the same BTC deposit", these inputs are intended to uniquely identify a Bitcoin deposit. However, the function does not validate `_depositTxid` and `_outputIndex`, allowing users to provide arbitrary values. This undermines the intended protection against deposit reuse.

Meanwhile, for valid `_depositTxid` and `_outputIndex` that can be verified by the minter, another qualified user can front-run a user's `addMintRequest` by sending a transaction with the same `_depositTxid` and `_outputIndex`. This would set `usedDepositTxs[depositKey]` to `true`, thereby blocking a legitimate `addMintRequest` that could have been verified as valid by the minter, and enabling a malicious DoS attack.

Additionally, the mint amount (`_amount`) is directly controlled by the user. Since the minting fee (`fee`) is fixed, users can potentially specify an unbounded `_amount` while paying only a constant fee. This opens the door to abuse, where users could mint excessive tokens at minimal cost.

```
RequestManager.Request memory r;
r.nonce = requestManager.nonce();
r.op = RequestManager.Operation.Mint;
r.status = RequestManager.Status.Pending;
r.srcChain = MAIN_CHAIN;
r.srcAddress = bytes(depositAddr);
r.dstChain = getSelfChainCode();
r.dstAddress = abi.encode(msg.sender);
r.amount = _amount;
r.extra = abi.encode(_depositTxid, _outputIndex);
r.fee = fee;

bytes32 hash = requestManager.addRequest(r);
_payFee(fee);
```

Recommendation

It is recommended to implement the following measures:

- Validation should be added to enforce uniqueness and authenticity of `_depositTxid` and `_outputIndex`.
- The `_amount` should be either verified against an off-chain proof of deposit or the fee should scale proportionally with the mint amount.

Alleviation

[KiKiFinance, 04/23/2025]: The team acknowledged the issue and decided to remain unchanged in the scope of the audit.

KII-18 | MISSING RECOVERY MECHANISM FOR BURNED TOKENS

Category	Severity	Location	Status
Logical Issue	Medium	contracts/iBTCwBridge.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 122~123	Resolved

Description

The `iBTCwBridge` contract allows qualified users to submit various types of requests, such as mint, burn, and cross-chain requests.

When users submit burn or cross-chain requests via the `addBurnRequest` and `addCrosschainRequest` functions, they are required to pay fees and burn `ibtcw` tokens from their own accounts:

```
IiBTCwToken(ibtcw).burn(msg.sender, r.amount);
```

However, the contract lacks any asset recovery mechanism. If a request cannot be processed as expected, the burned assets are irretrievable, potentially leading to permanent financial loss for the user.

Recommendation

It is recommended to implement a recovery mechanism that allows users to revoke their requests and reclaim their assets in cases where the requests cannot be processed.

Alleviation

[KiKiFinance, 04/23/2025]: The team heeded the advice and partially resolved the issue in commit `21f4681636cb4dcf4fb7366380f251259128e480`. The `revokeRequest` only mints back the burned tokens but does not refund the user's paid fees.

[CertiK, 04/27/2025]: The team has added a `revokeRequest` function that allows users to cancel pending requests and mint back their burned assets. This function relies on the following check:

```
    require(r.status == RequestManager.Status.Pending, "Not pending");
```

The `pending` status must be updated **before** the BTC is transferred to the user's designated withdrawal address.

Based on the current contract structure, only the `confirmBurnRequest` function, called by the `MINTER_ROLE`, updates the request status from `Pending` to `Confirmed`.

Since the BTC transfer process is outside the scope of this audit, we recommend the team carefully manage the execution of `confirmBurnRequest`, ensuring it is always invoked **before** the BTC transfer is completed.

If the request is processed on the destination chain but the status is **not** updated on this chain in time, it could **definitely** cause **double minting**.

[KiKiFinance, 04/28/2025]: The fee is to cover gas costs and is non-refundable.

KII-07 STATUS OF CURRENT ACCOUNT IN USER MANAGER NOT CHECKED WHEN MINTING TOKENS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/iBTCwBridge.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 219, 234	Resolved

Description

The `confirmCrosschainRequest()` and `confirmMintRequest()` functions can be called by the minter, and after verifying and updating the status of the corresponding request, tokens are minted to the specified `dstAddress`:

`iBTCwBridge.sol`

```
199     function confirmCrosschainRequest(RequestManager.Request memory r) external
onlyMinter whenNotPaused nonReentrant {
200         ...
201  @>     IiBTCwToken(ibtcw).mint(abi.decode(r.dstAddress, (address)), r.amount);
202     }
```

`iBTCwBridge.sol`

```
223     function confirmMintRequest(bytes32 _hash) external
onlyMinter whenNotPaused nonReentrant {
224         ...
225  @>     IiBTCwToken(ibtcw).mint(user, r.amount);
226     }
```

However, there is no check to determine whether the `dstAddress` is in a locked state in the `UserManager`.

This is especially problematic in `confirmCrosschainRequest()`, where a user can send tokens to a locked `dstAddress` through the cross-chain bridge.

Recommendation

It is recommended to add the corresponding checks and logic.

Alleviation

[KiKiFinance, 04/23/2025]: The team heeded the advice and resolved the issue by adding the following check in commit ce596da8e9ae68ec557304c504501adcc6a9c88d:

```
        address dst = abi.decode(r.dstAddress, (address));
        (bool locked, , ) = userManager.userInfo(dst);
        require(!locked, "UserManager: user locked");
```

KII-08 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/iBTCwBridge.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 58~63, 76	Resolved

Description

The cited address input is missing a check that it is not `address(0)`.

Recommendation

We recommend adding a check the passed-in address is not `address(0)` to prevent unexpected errors.

Alleviation

[KiKiFinance, 04/27/2025]: The team heeded the advice and resolved the issue by adding zero address validation in commit `21f4681636cb4dcf4fb7366380f251259128e480`.

KII-09 `addDstChains()` DOES NOT VALIDATE WHETHER THE DESTINATION CHAIN IS `block.chainId`

Category	Severity	Location	Status
Coding Issue	Minor	contracts/ChainManager.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 27~35	Resolved

Description

The function `addDstChains()` does not validate whether `_dstChains[i]` is equal to `block.chainId`. It should not allow adding `_dstChains[i]` to `dstChains` when it is equal to `block.chainId`. Otherwise, if `block.chainId` is added to the `dstChains` in the `ChainManager` contract, the `Bridge` contract would be allowed to perform a "fake" cross-chain transfer — effectively just transferring tokens on the same chain.

If combined with the vulnerability described in *Status of Current Account in User Manager Not Checked When Minting Tokens*, this could be exploited to transfer tokens to a locked account.

Recommendation

It is recommended to add a check in the function to ensure `_dstChains[i]` is not equal to `block.chainId`.

Alleviation

[KiKiFinance, 04/25/2025]: The team heeded the advice and resolved the issue by adding `currentChain` check in commit `21f4681636cb4dcf4fb7366380f251259128e480`.

KII-10 INACTIVE PAUSE FUNCTIONALITY IN BRIDGE AND REQUESTMANAGER CONTRACTS

Category	Severity	Location	Status
Design Issue	Minor	contracts/RequestManager.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 9; contracts/iBTCwBridge.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 19	Resolved

Description

The `Bridge` contract inherits from `PausableUpgradeable`, gaining access to the internal `_pause` and `_unpause` functions, and applies the `whenNotPaused` modifier to some of its functions. However, it does not expose any public or external methods to trigger these internal functions and update the `_paused` state, rendering the pause functionality ineffective.

Similarly, the `RequestManager` contract also inherits from `PausableUpgradeable`. However, it neither exposes any public or external methods to invoke the pause functions nor uses the `whenNotPaused` or `whenPaused` modifiers in its logic. As a result, the pausing mechanism has no practical impact on the contract's behavior.

Recommendation

It is recommended to either remove the `PausableUpgradeable` inheritance if the pause functionality is not needed, or fully implement the pausing mechanism by exposing public functions that call `_pause` and `_unpause`, and by incorporating the `whenNotPaused` and `whenPaused` modifiers where appropriate within the contract.

Alleviation

[KiKiFinance, 04/23/2025]: The team heeded the advice and resolved the issue by fully implementing the pausing mechanism in the `Bridge` contract and removing the inheritance of `PausableUpgradeable` from the `RequestManager` in commit `21f4681636cb4dcf4fb7366380f251259128e480`.

KII-15 CONFIRMATION ON HASH COLLISION RISKS AND BRIDGE DESIGN

Category	Severity	Location	Status
Logical Issue	Minor	contracts/iBTCwBridge.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 210~211	Acknowledged

Description

The `confirmCrosschainRequest` function is invoked by the *minter* role to confirm a cross-chain request. It relies on the `getCrossSourceRequestHash` method to regenerate the original hash computed on the source chain. The following fields are included in the hash:

```
r.op = RequestManager.Operation.CrosschainRequest
r.nonce
r.srcChain
r.srcAddr
r.dstChain = getSelfChainCode()
r.dstAddress
r.amount
r.extra = ""
r.fee
```

Notably, the hash does **not** include the bridge contract address or the current chain code. This raises concerns about potential hash collisions in scenarios where multiple bridge contracts are deployed on the same source chain, or when identical bridge contract addresses exist across different source chains.

If the same user submits two identical cross-chain requests on two different bridge contracts on the source chain, the resulting hash values (`srcHash`) could possibly be identical because the hashed fields, especially `r.srcChain` and `r.srcAddr` , remain the same. Consequently, only one of these requests will be accepted and recorded by the destination contract due to the following checks:

```
        require(r.getCrossSourceRequestHash() == srcHash, "Source request hash is incorrect");
        require(crosschainRequestConfirmation[srcHash] == bytes32(0), "Source request already confirmed");
```

This presents a risk of unintended request rejection or denial of service if multiple bridge contracts exist without proper disambiguation.

To properly assess the bridge mechanism and validate its design assumptions, we request clarification from the team on the intended deployment model for bridge contracts. Specifically:

- Is the bridge architecture designed as a **1-to-1**, **n-to-1**, or **n-to-n** model?
- How many bridge contracts are expected to be deployed per chain?

A clearly defined architecture is essential for reviewing and validating the robustness and correctness of the cross-chain request processing mechanism.

Recommendation

We recommend that the development team clarify the relevant mechanism and ensure it is properly enforced to prevent potential hash collision issues.

Alleviation

[KiKiFinance, 04/23/2025]: The bridge architecture is an N-to-N model, and only one bridge contract is deployed per chain.

KII-17 | MISSING INPUT VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/UserManager.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 36~37, 48~49; contracts/iBTCwBridge.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 109, 139~140	Resolved

Description

The `addQualifiedUser` function currently lacks validation to ensure that the `user`, `depositAddress`, and `withdrawalAddress` parameters are valid. If `user` is set to the zero address, the qualified user information becomes inaccessible. Similarly, if either `depositAddress` or `withdrawalAddress` is an empty value, the user will be unable to submit mint or burn requests due to the following checks:

```
require(bytes(depositAddr).length > 0, "User not qualified");
require(bytes(withdrawalAddr).length > 0, "User not qualified");
```

Both `depositAddress` and `withdrawalAddress` are also critical in constructing mint and burn requests. In the `addMintRequest` function, `depositAddr` is assigned to the `srcAddress` field of the request:

```
r.srcChain = MAIN_CHAIN;
r.srcAddress = bytes(depositAddr);
```

However, there is no validation to ensure that `depositAddr` represents a valid address on the source chain (`MAIN_CHAIN`).

Similarly, in the `addBurnRequest` function, `withdrawalAddr` is assigned to the `dstAddress` field:

```
r.dstChain = MAIN_CHAIN;
r.dstAddress = bytes(withdrawalAddr);
```

Yet again, the contract does not verify that `withdrawalAddr` is a valid destination address on `MAIN_CHAIN`.

The same issue exists in the `editQualifiedUser` function, where updated `depositAddress` and `withdrawalAddress` values are accepted without checks. These should also be validated to ensure they are proper addresses on `MAIN_CHAIN`.

Additionally, the `addCrosschainRequest` function allows a qualified user to specify a `_targetAddress` on a `_targetChain`. However, it does not perform any validation on `_targetAddress`. If a user mistakenly provides an invalid destination address, the associated assets will be irreversibly lost, as they are burned on the source chain and cannot be recovered:

```
r.dstChain = _targetChain;
r.dstAddress = _targetAddress;
```

Recommendation

It is recommended to implement comprehensive input validations to ensure all parameters, such as `user`, `depositAddress`, `withdrawalAddress`, and `_targetAddress` are valid values.

Alleviation

[KiKiFinance, 04/23/2025]:
Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/KikiFinance/ibtc.w-contract/commit/33fbc1ba4f67a4e322104c074a4b1933a7c45f72

[KiKiFinance, 04/23/2025]:
For EVM cross-chain transfers, we use addEVMCrosschainRequest, where targetAddress is of type address, which prevents supplying an incorrect address and losing funds.

As for depositAddress and withdrawalAddress, these are Bitcoin addresses. The contract currently has no reliable way to rigorously validate their correctness on-chain, so we rely on front-end validation and manual checks by the administrator during approval to ensure the addresses are valid.

[KiKiFinance, 04/23/2025]: cross chain check lock user

https://github.com/KikiFinance/ibtc.w-contract/commit/4cef7735ec040e7a152d02592592d4a818893881

[CertiK, 04/27/2025]:

- Simply checking if the address length is greater than zero is insufficient, as different chains may require addresses to meet specific formats.
    - Since user registration is handled by the `DEFAULT_ADMIN_ROLE`, we recommend the team pay close attention when setting critical parameters, to prevent unexpected user loss due to development errors.
- The `addCrosschainRequest` function allows users to specify a target address. If the target address is invalid, the user can only revoke the request but will lose the cross-chain fee.
    - We suggest either validating the address format or clearly informing users of this potential risk.
- In the latest commit, the following validation was removed:

```
(bool locked, string memory depositAddr, ) = userManager.userInfo(msg.sender);
require(bytes(depositAddr).length > 0, "User not qualified");
```

It now only checks the `locked` status:

```
(bool locked, , ) = userManager.userInfo(msg.sender);
```

- This allows unqualified users (not in `userInfo`) to submit cross-chain requests, as the default `locked` value would be `false`. We would like to confirm with the team whether this behavior aligns with the intended design.

[KiKiFinance, 04/28/2025]: The contract's validation is inflexible, so we'll perform validation on the front end and warn users of the risks. By default, unregistered users are allowed to initiate cross-chain requests.

KII-01 | CONFIRMATION ON OFF-CHAIN HANDLER

Category	Severity	Location	Status
Design Issue	Informational	contracts/iBTCwBridge.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 85, 122, 151, 199, 223, 238	Resolved

Description

To facilitate the review of the bridge contract, we would appreciate further clarification regarding the off-chain processes and certain implementation details. Specifically, we would like to understand the following:

1. **Mint Flow (`addMintRequest` → `confirmMintRequest`)**
   - How is the off-chain process managed from the moment a user submits a mint request until it is confirmed on the other chain?
   - What do the parameters `_depositTxid` and `_outputIndex` represent in this context, and what role do they play in ensuring the correctness or finality of the mint operation?
2. **Burn Flow (`addBurnRequest` → `confirmBurnRequest`)**
   - Similarly, how is the burn flow handled off-chain before reaching `confirmBurnRequest`?
   - Could you clarify the meaning and function of `_withdrawalTxid` and `_outputIndex` in this process?
3. **Cross-Chain Flow (`addCrosschainRequest` → `confirmCrosschainRequest`)**
   - How is the off-chain process managed for cross-chain requests, and what is the intended flow leading to the confirmation on the destination chain?
   - We noticed that the `confirmCrosschainRequest` function sets the status to `RequestManager.Status.Unused`. Is this the intended behavior, or should it be set to `Status.Confirmed` instead?
   - Since `confirmCrosschainRequest` uses `getCrossSourceRequestHash()` to verify the source hash, is there any off-chain component responsible for updating the status from `Pending` to `Unused`, and changing the operation from `CrosschainRequest` to `CrosschainConfirm`?
4. **Unused Enum Values**
   - Across the contract, we observed that `Status.Rejected` value is defined but never used. Could you clarify whether this is reserved for future use or was unintentionally left unused?

- If it is not part of the intended design, we recommend either removing it to simplify the codebase or implementing logic to make use of it where appropriate.

```
enum Operation { Nop, Mint, Burn, CrosschainRequest, CrosschainConfirm }
enum Status { Unused, Pending, Confirmed, Rejected }
```

Clarifying these points will help us better evaluate the robustness and completeness of the bridge logic, particularly around cross-chain coordination and off-chain reliability.

Recommendation

We would like the project team to clarify the reasoning behind this design and confirm whether this is indeed the intended behavior.

Alleviation

[KiKiFinance, 04/23/2025]:

Mint Flow (addMintRequest confirmMintRequest)

This project is a Wrapped BTC asset. Each whitelisted user is pre-assigned a unique BTC deposit address and a withdrawal address. After the user sends BTC to their designated deposit address, they call addMintRequest to initiate the mint. The _depositTxid parameter is the TxID of that BTC deposit transaction, and _outputIndex specifies the output index in that transaction corresponding to the user deposit address this precisely identifies the source of funds and ensures request uniqueness.

Burn Flow (addBurnRequest confirmBurnRequest)

When the user calls addBurnRequest, the system transfers the requested amount of BTC (minus on-chain fees) from the project custody address to the user designated withdrawal address.

The _withdrawalTxid parameter is the TxID of that BTC transfer transaction, and _outputIndex denotes the output index corresponding to the user withdrawal address used to verify the burned amount and target address.

Cross-Chain Flow (addCrosschainRequest confirmCrosschainRequest)

Off-chain services watch for burn events on Chain A, manually verify the burn, then submit the related parameters and their hash to Chain B. On Chain B, confirmCrosschainRequest is called to confirm the cross-chain request and mint the equivalent Wrapped BTC tokens. By design, every new request starts in Unused status and only changes after confirmation. When confirmCrosschainRequest is submitted, the corresponding request on Chain A is cleared; its status remains Unused to avoid duplicate processing.

Unused Enum Values

`Status.Rejected` was removed in commit `21f4681636cb4dcf4fb7366380f251259128e480` .

KII-12 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	contracts/iBTCwToken.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 41, 45	Resolved

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

[KiKiFinance, 04/23/2025]: The team heeded the advice and resolved the issue by adding event of blacklist in commit `21f4681636cb4dcf4fb7366380f251259128e480` .

KII-13 ADDING EXTRA DATA IN `confirmBurnRequest` WILL RESULT IN A DIFFERENT HASH FOR THE REQUEST

Category	Severity	Location	Status
Inconsistency	Informational	contracts/RequestManager.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 86; contracts/iBTCwBridge.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 256	Acknowledged

Description

In `confirmBurnRequest()`, the `extra` field of `requests[_hash]` is modified:

`iBTCwBridge.sol`

```
    function confirmBurnRequest(
        bytes32 _hash,
        bytes32 _withdrawalTxid,
        uint256 _outputIndex
    ) external onlyMinter whenNotPaused nonReentrant {
        ...

// Call confirmRequestWithExtra to update the extra field with the BTC withdrawal data and confirm the request.

@>      requestManager.confirmRequestWithExtra(_hash, _withdrawalTxData);
    }
```

`RequestManager.sol`

```
    function confirmRequestWithExtra(bytes32 _hash, bytes memory newExtra) external {
        require(msg.sender == bridge, "Only bridge can call confirmRequest");
        Request storage r = requests[_hash];
        require(r.status == Status.Pending, "Request not pending");
@>      r.extra = newExtra;
        r.status = Status.Confirmed;
        emit RequestConfirmed(_hash);
    }
```

This causes the hash value to differ from the original value obtained from `getRequestHash`:

`RequestManager.sol`

```
    function getRequestHash(
        RequestManager.Request memory self
    ) internal pure returns (bytes32) {
        return keccak256(
            abi.encode(
                self.op,
                self.nonce,
                self.srcChain,
                self.srcAddress,
                self.dstChain,
                self.dstAddress,
                self.amount,
                self.fee,
                self.extra
            )
        );
    }
```

Recommendation

It is recommended to exclude the `extra` field from the `getRequestHash` function.

Alleviation

[KiKiFinance, 04/23/2025]: The `extra` field contains vital data for example, the source chain request hash used during cross-chain confirmation. That hash is crucial for verifying the authenticity of the user request, and even after any administrative operations, the original user-issued request hash must be preserved.

This ensures that even after modifying the `extra` field, recomputing the hash will still yield the same result.

KII-16 CONFIRMATION ON THE STATUS OF CROSS-CHAIN REQUEST

Category	Severity	Location	Status
Volatile Code	Informational	contracts/iBTCwBridge.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 203~204	Acknowledged

Description

The `confirmCrosschainRequest()` function is called by the *minter* role to process a cross-chain request and mint tokens to the specified destination address. However, this function only checks whether the request status is `RequestManager.Status.Unused` prior to processing, but does not update the status afterward (e.g., to `Confirmed`).

Recommendation

We would like to discuss this behavior with the team to confirm whether it aligns with the intended design.

Alleviation

[KiKiFinance, 04/23/2025]: Because confirmCrosschainRequest is submitted off-chain by the minter using the CrosschainConfirm operation code and is intended to be a stateless request we enforce that its status must be Unused. This guarantees that only requests which the minter has already validated and filtered off-chain can be submitted on-chain, preventing the minter from accidentally pushing an arbitrary user request into the contract.

Issue acknowledged. I won't make any changes for the current version.

KII-19 | CONFIRMATION ON `addMintRequest()`

Category	Severity	Location	Status
Design Issue	Informational	contracts/iBTCwBridge.sol (3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a): 85	Acknowledged

Description

The `addMintRequest` function allows a qualified user to submit a mint request. However, it sets `srcChain` to `MAIN_CHAIN` and `srcAddress` to `depositAddr`:

```
        r.srcChain = MAIN_CHAIN;
        r.srcAddress = bytes(depositAddr);
        r.dstChain = getSelfChainCode();
        r.dstAddress = abi.encode(msg.sender);
```

This behavior appears counterintuitive. The request is being submitted from the current chain, yet it specifies a different chain (MAIN_CHAIN) as the source, while the destination chain is the current chain.

Recommendation

We'd like to confirm whether this design is intentional, or if it's a potential misconfiguration.

Alleviation

[KiKiFinance, 04/24/2025]: srcChain and dstChain represent the source and destination chains respectively. In the context of a mint request, it's analogous to the user bridging BTC from the Bitcoin chain to the EVM chain, and here MAIN_CHAIN refers to the Bitcoin chain.

Issue acknowledged. I won't make any changes for the current version.

FORMAL VERIFICATION | KIKIFINANCE IBTC.W

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of contracts derived from AccessControl v4.4

We verified properties of the public interface of contracts that provide an AccessControl-v4.4 compatible API. This involves:

- The `hasRole` function, which returns `true` if an account has been granted a specific `role`.
- The `getRoleAdmin` function, which returns the admin role that controls a specific `role`.
- The `grantRole` and `revokeRole` functions, which are used for granting a `role` to an account and revoking a `role` from an `account`, respectively.
- The `renounceRole` function, which allows the calling account to revoke a `role` from itself.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
accesscontrol-renouncerole-revert-not-sender	`renounceRole` Reverts When Caller Is Not the Confirmation Address
accesscontrol-default-admin-role	AccessControl Default Admin Role Invariance
accesscontrol-renouncerole-succeed-role-renouncing	`renounceRole` Successfully Renounces Role
accesscontrol-getroleadmin-succeed-always	`getRoleAdmin` Function Always Succeeds
accesscontrol-hasrole-succeed-always	`hasRole` Function Always Succeeds
accesscontrol-getroleadmin-change-state	`getRoleAdmin` Function Does Not Change State
accesscontrol-hasrole-change-state	`hasRole` Function Does Not Change State
accesscontrol-grantrole-correct-role-granting	`grantRole` Correctly Grants Role
accesscontrol-revokerole-correct-role-revoking	`revokeRole` Correctly Revokes Role

Verification of Pausable ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the pausable ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply`, which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Transfers
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-fail-recipient-overflow	`transferFrom` Prevents Overflows in the Recipient's Balance
erc20-transfer-recipient-overflow	`transfer` Prevents Overflows in the Recipient's Balance
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20-transfer-never-return-false	`transfer` Never Returns `false`
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-transferfrom-revert-zero-argument	`transferFrom` Fails for Transfers with Zero Address Arguments

Property Name	Title
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-approve-succeed-normal	`approve` Succeeds for Valid Inputs
erc20pausable-transferfrom-revert-paused	`transferFrom` Fails for a Paused Contract
erc20pausable-transfer-revert-paused	`transfer` Fails for a Paused Contract
erc20-approve-false	If `approve` Returns `false`, the Contract's State Is Unchanged
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-approve-never-return-false	`approve` Never Returns `false`
erc20-transferfrom-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-transfer-false	If `transfer` Returns `false`, the Contract State Is Not Changed
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Transfers

Verification Results

For the following contracts, formal verification established that each of the properties that were in scope of this audit (see scope) are valid:

Detailed Results For Contract ChainManager (contracts/ChainManager.sol) In Commit 21f4681636cb4dcf4fb7366380f251259128e480

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	● True	
accesscontrol-renouncerole-succeed-role-renouncing	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	● True	
accesscontrol-hasrole-change-state	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results For Contract iBTCwBridge (contracts/iBTCwBridge.sol) In Commit 21f4681636cb4dcf4fb7366380f251259128e480

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-change-state	● True	
accesscontrol-hasrole-succeed-always	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-change-state	● True	
accesscontrol-getroleadmin-succeed-always	● True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	● True	
accesscontrol-renouncerole-succeed-role-renouncing	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results For Contract FeeConfigStore (contracts/FeeConfigStore.sol) In Commit 21f4681636cb4dcf4fb7366380f251259128e480

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	● True	
accesscontrol-hasrole-change-state	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-succeed-role-renouncing	● True	
accesscontrol-renouncerole-revert-not-sender	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results For Contract UserManager (contracts/UserManager.sol) In Commit 21f4681636cb4dcf4fb7366380f251259128e480

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	● True	
accesscontrol-renouncerole-succeed-role-renouncing	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-change-state	● True	
accesscontrol-hasrole-succeed-always	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results For Contract RequestManager (contracts/RequestManager.sol) In Commit 21f4681636cb4dcf4fb7366380f251259128e480

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	● True	
accesscontrol-renouncerole-succeed-role-renouncing	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	● True	
accesscontrol-hasrole-change-state	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results For Contract ChainManager (contracts/ChainManager.sol) In Commit 3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	● True	
accesscontrol-hasrole-change-state	● True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	● True	
accesscontrol-renouncerole-succeed-role-renouncing	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results For Contract FeeConfigStore (contracts/FeeConfigStore.sol) In Commit 3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	● True	
accesscontrol-renouncerole-succeed-role-renouncing	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	● True	
accesscontrol-hasrole-change-state	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results For Contract RequestManager (contracts/RequestManager.sol) In Commit 3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	● True	
accesscontrol-renouncerole-succeed-role-renouncing	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	● True	
accesscontrol-hasrole-change-state	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results For Contract UserManager (contracts/UserManager.sol) In Commit 3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	True	
accesscontrol-hasrole-change-state	True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	True	
accesscontrol-renouncerole-succeed-role-renouncing	True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-change-state	True	
accesscontrol-getroleadmin-succeed-always	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results For Contract iBTCwBridge (contracts/iBTCwBridge.sol) In Commit 3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	● True	
accesscontrol-renouncerole-succeed-role-renouncing	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	● True	
accesscontrol-hasrole-change-state	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-change-state	True	
accesscontrol-getroleadmin-succeed-always	True	

In the remainder of this section, we list all contracts where formal verification of at least one property was not successful. There are several reasons why this could happen:

- False: The property is violated by the project.
- Inconclusive: The proof engine cannot prove or disprove the property due to timeouts or exceptions.
- Inapplicable: The property does not apply to the project.

Detailed Results For Contract iBTCwToken (contracts/iBTCwToken.sol) In Commit 21f4681636cb4dcf4fb7366380f251259128e480

Verification of Pausable ERC-20 Compliance

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-exceed-balance	True	
erc20-transfer-recipient-overflow	Inconclusive	
erc20-transfer-never-return-false	True	
erc20pausable-transfer-revert-paused	True	
erc20-transfer-false	True	
erc20-transfer-revert-zero	True	
erc20-transfer-correct-amount	True	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-change-state	True	
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-fail-recipient-overflow	Inconclusive	
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-revert-zero-argument	True	
erc20pausable-transferfrom-revert-paused	True	
erc20-transferfrom-false	True	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-change-state	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-succeed-always	True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-change-state	True	
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-correct-amount	True	
erc20-approve-succeed-normal	True	
erc20-approve-false	True	
erc20-approve-never-return-false	True	

Detailed Results For Contract iBTCwToken (contracts/iBTCwToken.sol) In Commit 3e2a3a79cdc7513eef22aeb9b7e7d3cf27a7283a

Verification of Pausable ERC-20 Compliance

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	
erc20-allowance-change-state	True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-false	True	
erc20-approve-never-return-false	True	
erc20-approve-succeed-normal	True	
erc20-approve-revert-zero	True	
erc20-approve-correct-amount	True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-false	True	
erc20pausable-transferfrom-revert-paused	True	
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-revert-zero-argument	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-fail-recipient-overflow	Inconclusive	

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20pausable-transfer-revert-paused	True	
erc20-transfer-never-return-false	True	
erc20-transfer-false	True	
erc20-transfer-revert-zero	True	
erc20-transfer-correct-amount	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-recipient-overflow	Inconclusive	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	

APPENDIX | KIKIFINANCE IBTC.W

Finding Categories

Categories	Description
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.

- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed AccessControl-v4.4 Properties

Properties related to function `renounceRole`

accesscontrol-renouncerole-revert-not-sender

The `renounceRole` function must revert if the caller is not the same as `account`.

Specification:

```
reverts_when account != msg.sender;
```

accesscontrol-renouncerole-succeed-role-renouncing

After execution, `renounceRole` must ensure the caller no longer has the renounced role.

Specification:

```
ensures !hasRole(role, account);
```

Properties related to function `DEFAULT_ADMIN_ROLE`

accesscontrol-default-admin-role

The default admin role must be invariant, ensuring consistent access control management.

Specification:

```
invariant DEFAULT_ADMIN_ROLE() == 0x00;
```

Properties related to function `getRoleAdmin`

accesscontrol-getroleadmin-change-state

The `getRoleAdmin` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-getroleadmin-succeed-always

The `getRoleAdmin` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `hasRole`

accesscontrol-hasrole-change-state

The `hasRole` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-hasrole-succeed-always

The `hasRole` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `grantRole`

accesscontrol-grantrole-correct-role-granting

After execution, `grantRole` must ensure the specified account has the granted role.

Specification:

```
ensures hasRole(role, account);
```

Properties related to function `revokeRole`

accesscontrol-revokerole-correct-role-revoking

After execution, `revokeRole` must ensure the specified account no longer has the revoked role.

Specification:

```
ensures !hasRole(role, account);
```

Description of the Analyzed ERC-20-Pausable Properties

Properties related to function `transfer`

erc20-transfer-correct-amount

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

Specification:

```
requires recipient != msg.sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(recipient) == \old(balanceOf(recipient) + amount)
&& balanceOf(msg.sender) == \old(balanceOf(msg.sender) - amount);
  also
requires recipient == msg.sender;
ensures \result ==> balanceOf(msg.sender) == \old(balanceOf(msg.sender));
```

erc20-transfer-exceed-balance

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

Specification:

```
requires amount > balanceOf(msg.sender);
ensures !\result;
```

erc20-transfer-false

If the `transfer` function in contract `iBTCwToken` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transfer-never-return-false

The transfer function must never return `false` to signal a failure.

Specification:

```
ensures \result;
```

erc20-transfer-recipient-overflow

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

Specification:

```
requires recipient != msg.sender;
requires balanceOf(recipient) + amount > type(uint256).max;
ensures !\result;
```

erc20-transfer-revert-zero

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

Specification:

```
ensures \old(recipient) == address(0) ==> !\result;
```

erc20pausable-transfer-revert-paused

Any invocation of `transfer(recipient, amount)` must fail if the contract is paused.

Specification:

```
reverts_when paused();
```

Properties related to function `allowance`

erc20-allowance-change-state

Function `allowance` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-allowance-correct-value

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

Specification:

```
ensures \result == allowance(\old(owner), \old(spender));
```

erc20-allowance-succeed-always

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `transferFrom`

erc20-transferfrom-correct-allowance

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount`.

Specification:

```
ensures \result ==> allowance(\old(sender), msg.sender) == \old(allowance(sender,
msg.sender)) - \old(amount)
                  || (allowance(\old(sender), msg.sender) == \old(allowance(sender,
msg.sender)) && \old(allowance(sender, msg.sender)) == type(uint256).max);
```

erc20-transferfrom-correct-amount

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest`.

Specification:

```
requires recipient != sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient) +
amount)
                  && balanceOf(\old(sender)) == \old(balanceOf(sender) - amount);
 also
requires recipient == sender;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient));
```

erc20-transferfrom-fail-exceed-allowance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

Specification:

```
requires msg.sender != sender;
requires amount > allowance(sender, msg.sender);
ensures !\result;
```

erc20-transferfrom-fail-exceed-balance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

Specification:

```
requires amount > balanceOf(sender);
ensures !\result;
```

erc20-transferfrom-fail-recipient-overflow

Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail.

Specification:

```
requires recipient != sender;
requires balanceOf(recipient) + amount > type(uint256).max;
ensures !\result;
```

erc20-transferfrom-false

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transferfrom-never-return-false

The `transferFrom` function must never return `false`.

Specification:

```
ensures \result;
```

erc20-transferfrom-revert-zero-argument

All calls of the form `transferFrom(from, dest, amount)` must fail for transfers from or to the zero address.

Specification:

```
ensures \old(sender) == address(0) ==> !\result;
also
ensures \old(recipient) == address(0) ==> !\result;
```

erc20pausable-transferfrom-revert-paused

Any call of the form `transferFrom(from, dest, amount)` must fail for a paused contract.

Specification:

```
reverts_when paused();
```

Properties related to function `totalSupply`

erc20-totalsupply-change-state

The `totalSupply` function in contract iBTCwToken must not change any state variables.

Specification:

```
assignable \nothing;
```

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract iBTCwToken.

Specification:

```
ensures \result == totalSupply();
```

erc20-totalsupply-succeed-always

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `balanceOf`

erc20-balanceof-change-state

Function `balanceOf` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-balanceof-correct-value

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner`.

Specification:

```
ensures \result == balanceOf(\old(account));
```

erc20-balanceof-succeed-always

Function `balanceOf` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `approve`

erc20-approve-correct-amount

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`.

Specification:

```
requires spender != address(0);
ensures \result ==> allowance(msg.sender, \old(spender)) == \old(amount);
```

erc20-approve-false

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-approve-never-return-false

The function `approve` must never returns `false`.

Specification:

```
ensures \result;
```

erc20-approve-revert-zero

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

Specification:

```
ensures \old(spender) == address(0) ==> !\result;
```

erc20-approve-succeed-normal

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

Specification:

```
requires spender != address(0);
ensures \result;
reverts_only_when false;
```

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Entire Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.