cNGN stablecoin update

Security Assessment

CertiK Assessed on May 22nd, 2025

CertiK Assessed on May 22nd, 2025

cNGN stablecoin update

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
StableCoin	Ethereum (ETH)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 05/22/2025	N/A

CODEBASE	COMMITS
Base	a3593071a40a058354753c6a0d7cb5c10a87a3e6
Update1	b5ffb7e69146dd33f6a1eae5ba7b9e9ff69da958
Update2	df82ba1d3a6837403fc649689a6b276adfb2bf2f
View All in Codebase Page	View All in Codebase Page

Vulnerability Summary

18	14	2	1	1	0
Total Findings	Resolved	Multi-Sig	Partially Resolved	Acknowledged	Declined

2	Centralization	2 Multi-Sig	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.
1	Major	1 Resolved	Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
2	Medium	2 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
6	Minor	4 Resolved, 1 Partially 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.
7	Informational	7 Resolved	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 | CNGN STABLECOIN UPDATE

CODEBASE | CNGN STABLECOIN UPDATE

Repository

Base

Update1

Update2

Update3

Commit

a3593071a40a058354753c6a0d7cb5c10a87a3e6

b5ffb7e69146dd33f6a1eae5ba7b9e9ff69da958

df82ba1d3a6837403fc649689a6b276adfb2bf2f

f752865e8b38b391572782b40af0826786a29992

AUDIT SCOPE | CNGN STABLECOIN UPDATE

3 files audited ● 1 file with Acknowledged findings ● 2 files with Resolved findings

ID	Repo	File	SHA256 Checksum
● CN2	wrappedcbdc/stablecoin-cngn	contracts/Cngn2.sol	1ff94aa3ef999e67a6cde63a878b296a6c126d4ec066c2aef9075066dc3690d7
● OPE	wrappedcbdc/stablecoin-cngn	contracts/Operations2.sol	d9f2ad911beef9ce4c72e3264b2b16fde5418708d3f732eff11fd4d945cf4abd
● MUL	wrappedcbdc/stablecoin-cngn	contracts/Multisig.sol	65a68729963d324567a366f33f37310c0d467ed2c52d7b7f3bf50be85bcd2f09

APPROACH & METHODS | CNGN STABLECOIN UPDATE

This report has been prepared for WrappedCBDC to discover issues and vulnerabilities in the source code of the cNGN stablecoin update 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 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.

FINDINGS | CNGN STABLECOIN UPDATE

18	0	2	1	2	6	7
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for cNGN stablecoin update. Through this audit, we have uncovered 18 issues ranging from different severity levels. Utilizing the techniques of Static Analysis & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
CSU-07	**Centralized Control Of Contract Upgrade**	**Centralization**	**Centralization**	**2/2 Multi-Sig**
CSU-08	**Centralization Related Risks**	**Centralization**	**Centralization**	**2/2 Multi-Sig**
CSU-09	`approveTransaction()` Allows An Owner To Approve A Transaction Multiple Times	Logical Issue, Access Control	Major	Resolved
CSU-10	Signer Can Remove Other Approvals	Logical Issue, Access Control, Design Issue	Medium	Resolved
CSU-21	`removeOwner()` Does Not Remove Previous Approvals	Logical Issue	Medium	Resolved
CSU-11	Missing Zero Address Validation	Volatile Code	Minor	Partially Resolved
CSU-12	Inconsistency Between Transfer And TransferFrom Logic	Inconsistency	Minor	Acknowledged
CSU-13	Check-Effect-Interaction Pattern Violated	Concurrency	Minor	Resolved
CSU-22	`revokeApproval()` On An Expired Transaction	Inconsistency	Minor	Resolved
CSU-23	Potential `required` Update Issue	Logical Issue	Minor	Resolved

ID	Title	Category	Severity	Status
CSU-26	Inconsistency In `Multisig`	Inconsistency	Minor	Resolved
CSU-04	Information Needed On Upgrade Handling	Coding Issue	Informational	Resolved
CSU-14	Array Missing `pop` Function	Volatile Code	Informational	Resolved
CSU-15	Comparison To Boolean Constant	Coding Style	Informational	Resolved
CSU-16	Missing Emit Events	Coding Style	Informational	Resolved
CSU-17	Missing Error Message	Coding Style	Informational	Resolved
CSU-19	Commented Code	Coding Style	Informational	Resolved
CSU-24	`executeTransaction()` Is Set As `internal`	Design Issue	Informational	Resolved

CSU-07 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	Centralization	contracts/Operations.sol (Base): 9; contracts/cngn.sol (Base): 14; tron-contract/contracts/Operations.sol (Base): 9	2/2 Multi-Sig

Description

In the contract `Operations2` , the role `owner` has the authority to update the implementation.

The contract `Cngn2` is upgradable, therefore, a centralized role has the authority to update the implementation contract.

Any compromise of the privileged account may allow a hacker to take advantage of this authority and change the implementation logic, 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

[CertiK, 05/22/2025]:

Base Sepolia testnet - Block Height: 26074651 - Timestamp: May-22-2025 12:39:50 PM +UTC

`Operations2`

- **Implementation Contract:** `0xc06d842c4877ead9315357aba8ecdb7c94ccb7f5`
- **Proxy Contract:** `0xB88a696bf420D447e0eED3c9fB4851863cd50389`
- **Ownership Transfer Transaction:** `0x45184431fbbeebd13b70367b11726546a2565548270a2102a5e27f347748f5ec`
  Ownership has been transferred to the following Safe multisig wallet:
- **Safe Multisig Wallet:** `0xff93447E9b974041315496cbdCaAFEA7E2b0CC9b` (Base Sepolia)
- **Signers (2-of-2 multisig):**
  - `0xc5422E03B8250917023501E4697c738E7427b540` — Externally Owned Account (EOA)
  - `0xFA7894a527E564C4a4C8308631EB59aDCbBdD71a` — Externally Owned Account (EOA)

> The multisig is configured with a 2-of-2 signature requirement, meaning both signers must approve each transaction.

`Cngn2`

- **Implementation Contract:** `0xb300895b10dde01b60a474045157d3da0622539a`
- **Proxy Contract:** `0x5c9b9f716B84984e06c78da32bD70A75Df96C006`
- **Ownership Transfer Transaction:** `0x03f01ec7c4b9ca7f0522415f0e659a27bf4eae055e4e97cf62917f7ca6b185b1`
  Ownership has been transferred to the following Safe multisig wallet:
- **Safe Multisig Wallet:** `0xff93447E9b974041315496cbdCaAFEA7E2b0CC9b` (Base Sepolia)
- **Signers (2-of-2 multisig):**
  - `0xc5422E03B8250917023501E4697c738E7427b540` — Externally Owned Account (EOA)
  - `0xFA7894a527E564C4a4C8308631EB59aDCbBdD71a` — Externally Owned Account (EOA)

> The multisig is configured with a 2-of-2 signature requirement, meaning both signers must approve each transaction.

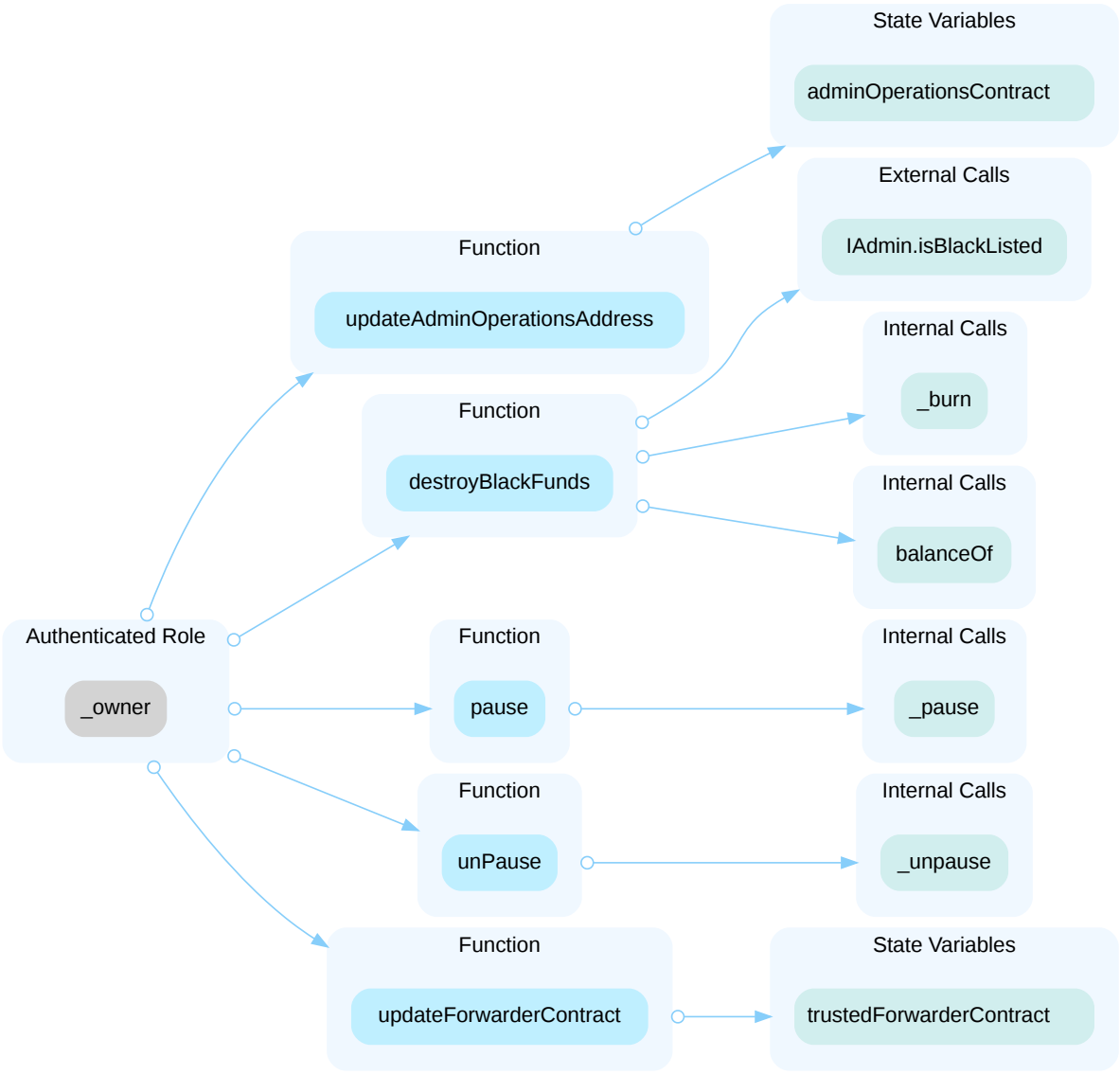
CSU-08 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	Centralization	contracts/Cngn2.sol (Base): 43, 50, 92, 99, 295, 300, 398; contracts/Operations2.sol (Base): 70, 84, 93, 103, 112, 124, 133, 147, 157, 166, 175, 185, 194, 202, 215	2/2 Multi-Sig

Description

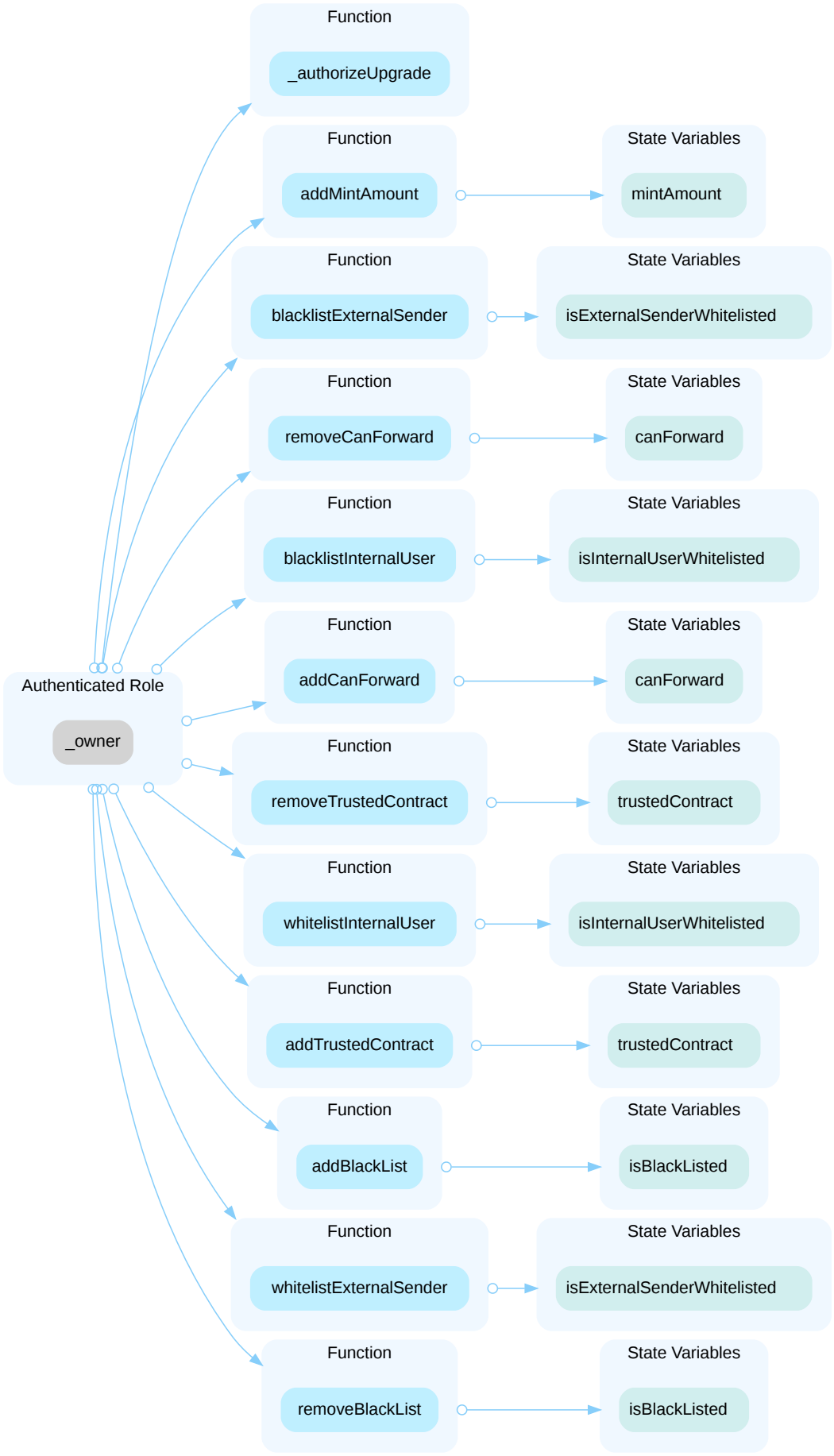
Cngn2

In the contract `Cngn2`, the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and pause or unpause the contract, update the admin operations contract address, update the trusted forwarder contract address, remove blacklisted user's funds.

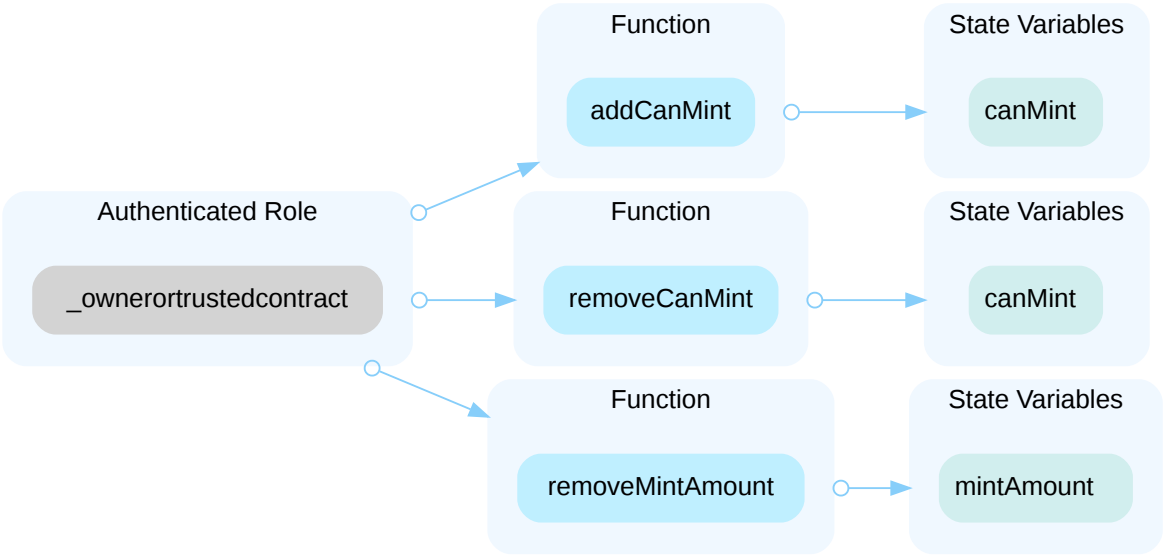


Operations2

In the contract `Admin2`, the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and whitelist or blacklist internal users, add or remove mint amounts for users, add or remove trusted contracts, whitelist or blacklist external senders, add or remove a user's forwarder status, and authorize contract upgrade implementations.


Function
_authorizeUpgrade
Function
addMintAmount
State Variables
mintAmount
Function
blacklistExternalSender
State Variables
isExternalSenderWhitelisted
Function
removeCanForward
State Variables
canForward
Function
blacklistInternalUser
State Variables
isInternalUserWhitelisted
Function
addCanForward
State Variables
canForward
Authenticated Role
_owner
Function
removeTrustedContract
State Variables
trustedContract
Function
whitelistInternalUser
State Variables
isInternalUserWhitelisted
Function
addTrustedContract
State Variables
trustedContract
Function
addBlackList
State Variables
isBlackListed
Function
whitelistExternalSender
State Variables
isExternalSenderWhitelisted
Function
removeBlackList
State Variables
isBlackListed

In the contract `Admin2`, the role `_ownerortrustedcontract` has authority over the functions shown in the diagram below. Any compromise to the `_ownerortrustedcontract` account may allow the hacker to take advantage of this authority and add users to the can mint list, remove the minting ability from a user or set the minting amount for a specific user to zero



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

[CertiK, 05/22/2025]:

Base Sepolia testnet - Block Height: 26074651 - Timestamp: May-22-2025 12:39:50 PM +UTC

`Operations2`

- **Implementation Contract:** `0xc06d842c4877ead9315357aba8ecdb7c94ccb7f5`
- **Proxy Contract:** `0xB88a696bf420D447e0eED3c9fB4851863cd50389`
- **Ownership Transfer Transaction:** `0x45184431fbbeebd13b70367b11726546a2565548270a2102a5e27f347748f5ec`
  Ownership has been transferred to the following Safe multisig wallet:
- **Safe Multisig Wallet:** `0xff93447E9b974041315496cbdCaAFEA7E2b0CC9b` (Base Sepolia)
- **Signers (2-of-2 multisig):**
  - `0xc5422E03B8250917023501E4697c738E7427b540` — Externally Owned Account (EOA)
  - `0xFA7894a527E564C4a4C8308631EB59aDCbBdD71a` — Externally Owned Account (EOA)

> The multisig is configured with a 2-of-2 signature requirement, meaning both signers must approve each transaction.

`Cngn2`

- **Implementation Contract:** `0xb300895b10dde01b60a474045157d3da0622539a`

- **Proxy Contract:** `0x5c9b9f716B84984e06c78da32bD70A75Df96C006`
- **Ownership Transfer Transaction:** `0x03f01ec7c4b9ca7f0522415f0e659a27bf4eae055e4e97cf62917f7ca6b185b1`
  Ownership has been transferred to the following Safe multisig wallet:
- **Safe Multisig Wallet:** `0xff93447E9b974041315496cbdCaAFEA7E2b0CC9b` (Base Sepolia)
- **Signers (2-of-2 multisig):**
  - `0xc5422E03B8250917023501E4697c738E7427b540` — Externally Owned Account (EOA)
  - `0xFA7894a527E564C4a4C8308631EB59aDCbBdD71a` — Externally Owned Account (EOA)

> The multisig is configured with a 2-of-2 signature requirement, meaning both signers must approve each transaction.

CSU-09 `approveTransaction()` ALLOWS AN OWNER TO APPROVE A TRANSACTION MULTIPLE TIMES

Category	Severity	Location	Status
Logical Issue, Access Control	Major	contracts/Multisig.sol (Base): 72~73	Resolved

Description

The `approveTransaction()` function currently lets a single owner register multiple approvals for the same transaction. This effectively allows one owner to increment the confirmation count at will—undermining the multisig's security model by bypassing the requirement for distinct owner endorsements before execution.

Recommendation

We recommend that each owner's approval for a given transaction be recorded only once, preventing any single owner from casting duplicate confirmations.

Alleviation

[CertiK, 05/12/2025]: The team heeded the advice and resolved the issue in commit b5ffb7e69146dd33f6a1eae5ba7b9e9ff69da958.

[CertiK, 05/22/2025]: The team addressed the issue by discontinuing their custom multisig implementation and opting to use a Safe multisig contract. Commit: 91c9cee976df029407030bbaec5a40093a64e9ab.

CSU-10 | SIGNER CAN REMOVE OTHER APPROVALS

Category	Severity	Location	Status
Logical Issue, Access Control, Design Issue	Medium	contracts/Multisig.sol (Base): 98~99	Resolved

Description

In `Multisig`, the function `rejectTransaction()` allows any owner to set `approvals` for a specific pending transaction to zero.

This means an owner can prevent any transaction from being executed by consistently resetting the approval to zero.

Recommendation

We recommend rewriting this function to enforce the rule that an owner can only remove their own approval from an existing transaction.

Alleviation

[CertiK, 05/12/2025]: The team heeded the advice and resolved the issue in commit b5ffb7e69146dd33f6a1eae5ba7b9e9ff69da958.

[CertiK, 05/22/2025]: The team addressed the issue by discontinuing their custom multisig implementation and opting to use a Safe multisig contract. Commit: 91c9cee976df029407030bbaec5a40093a64e9ab.

CSU-21 `removeOwner()` DOES NOT REMOVE PREVIOUS APPROVALS

Category	Severity	Location	Status
Logical Issue	Medium	Multisig.sol (Update1): 354~355, 387~391	Resolved

Description

The functions `Multisig.removeOwner` and `Multisig.replaceOwner()` do not erase the previous approvals on active transactions made by the removed owner.
As a result, transactions may retain outdated approvals, potentially leading to execution with more approvals than the current set of valid owners.

Recommendation

We recommend ensuring that active transaction approvals are correctly updated when an owner is removed or changed.

Alleviation

[CertiK, 05/15/2025]: The team heeded the advice and resolved the issue in commit dddd97c1f7235460655e18191b9f2282bd2c7ade.

[CertiK, 05/22/2025]: The team addressed the issue by discontinuing their custom multisig implementation and opting to use a Safe multisig contract. Commit: 91c9cee976df029407030bbaec5a40093a64e9ab.

CSU-11 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/Cngn2.sol (Base): 66, 67, 95, 102	Partially 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

[WrappedCBDC, 05/15/2025]: We have added the check to functions that are necessary.

https://github.com/wrappedcbdc/stablecoin-cngn/commit/983f2827b52e4cfb18bd733463c58716d5359c41

[CertiK, 05/15/2025]: The team heeded the advice and partially resolved the issue in commit 983f2827b52e4cfb18bd733463c58716d5359c41.

CSU-12 INCONSISTENCY BETWEEN TRANSFER AND TRANSFERFROM LOGIC

Category	Severity	Location	Status
Inconsistency	Minor	contracts/Cngn2.sol (Base): 157; Cngn2.sol (Update1): 196; cngn.sol (Update1): 198	Acknowledged

Description

The `transfer()` function contains different restrictions or additional logic compared to the `transferFrom()` function, resulting in inconsistent behavior between these two core ERC-20 functions. This inconsistency can lead to a situation where a transfer might fail when using `transfer()` due to the imposed restrictions, but the same transfer could succeed using `transferFrom()`, potentially bypassing the intended security checks or business logic.

Recommendation

To ensure consistent behavior between `transfer()` and `transferFrom()`, apply the following steps:

1. **Harmonize Function Logic**: Review and align the restrictions and checks in both `transfer()` and `transferFrom()` functions so that they enforce the same rules.
2. **Reevaluate Business Logic**: Assess the necessity of the restrictions in the `transfer()` function. If they are essential for the token's integrity and security, ensure that the same logic is implemented in the `transferFrom()` function.
3. **Thorough Testing**: Test both functions extensively to ensure that they behave consistently under various scenarios, including edge cases.

Alleviation

[WrappedCBDC, 05/09/2025]: The necessary checks are in place already, which is the blacklisting on both transfer and transferFrom. Other check on the transfer is for internal system working tool.

CSU-13 | CHECK-EFFECT-INTERACTION PATTERN VIOLATED

Category	Severity	Location	Status
Concurrency	Minor	contracts/Cngn2.sol (Base): 246~278, 335, 337; contracts/Multisig.sol (Base): 84~95	Resolved

Description

The Checks–Effects–Interactions pattern is a development guideline that recommends, in each function, to first perform all input validations (checks), then update the contract's state (effects), and only afterwards execute any external calls (interactions). By following this sequence, you ensure state changes are committed before control leaves the contract, which reduces the risk of reentrancy attacks and makes the contract's behavior easier to reason about.

Multisig

External call(s)

```
91          (bool success, ) = txn.to.call{value: txn.value}(txn.data);
```

Events emitted after the call(s)

```
94          emit TransactionExecuted(msg.sender, transactionId);
```

Emitting the event after the external call violates the checks-effects-interactions pattern.

Cngn2

External call(s)

```
266         require(
267             IAdmin(adminOperationsContract).removeCanMint(signer) == true,
268             "Failed to revoke minting authorization"
269         );
```

```
270         require(
271             IAdmin(adminOperationsContract).removeMintAmount(signer) == true,
272             "Failed to remove mint amount"
273         );
```

State variables written after the call(s)

```
275          _mint(_mintTo, _amount);
```

- This function call executes the following assignment(s).
- In `Cngn2._mint`,
    - `_totalSupply += amount`

Updating `_totalSupply` after the external call to `adminOperationsContract` violates the checks-effects-interactions pattern.

Recommendation

We recommend adhering to the Checks-Effects-Interactions Pattern to follow best practice.

In `Multisig`, the event should be emitted before the external call:

```
....
        emit TransactionExecuted(msg.sender, transactionId);
....
        (bool success, ) = txn.to.call{value: txn.value}(txn.data);
....
```

In `Cngn2`, the `_mint()` call should take place before the external call:

```
....
        _mint(_mintTo, _amount);
....
        require(
            IAdmin(adminOperationsContract).removeCanMint(signer) == true,
            "Failed to revoke minting authorization"
        );

        require(
            IAdmin(adminOperationsContract).removeMintAmount(signer) == true,
            "Failed to remove mint amount"
        );
....
```

Alleviation

[CertiK, 05/15/2025]: The team heeded the advice and resolved the issue in commit 03748fd1ca9c04891c5c72f9ed69bcaf5d64721a.

[CertiK, 05/22/2025]: The team addressed the issue by discontinuing their custom multisig implementation and opting to use a Safe multisig contract. Commit: 91c9cee976df029407030bbaec5a40093a64e9ab.

CSU-22 | `revokeApproval()` ON AN EXPIRED TRANSACTION

Category	Severity	Location	Status
Inconsistency	Minor	Multisig.sol (Update1): 228~229	Resolved

Description

`Multisig.revokeApproval()` does not prevent revoking an expired past approval on a transaction.

Recommendation

We recommend preventing modifications to parameters on expired transactions to enhance consistency.

Alleviation

[CertiK, 05/15/2025]: The team heeded the advice and resolved the issue in commit 739650f5cc22fbbc7041c0958ada7ac22b3cb312.

[CertiK, 05/22/2025]: The team addressed the issue by discontinuing their custom multisig implementation and opting to use a Safe multisig contract. Commit: 91c9cee976df029407030bbaec5a40093a64e9ab.

CSU-23 | POTENTIAL `required` UPDATE ISSUE

Category	Severity	Location	Status
Logical Issue	Minor	Multisig.sol (Update1): 266~270, 415~416	Resolved

Description

`Multisig.changeRequirement()` can set `required` to a value lower than the current transaction's approval. As a result, the `execute()` function will revert for those transactions even if they have enough voters since `executeTransaction()` requires `approvalCount[transactionId] == required`.

This would require canceling those transactions and recreating them.

Recommendation

We recommend rewriting contract logic to prevent this issue.

Alleviation

[CertiK, 05/15/2025]: The team heeded the advice and resolved the issue in commit 26a33ae0a1fd263bad8cbc7c21bb50ca52a4931b.

[CertiK, 05/22/2025]: The team addressed the issue by discontinuing their custom multisig implementation and opting to use a Safe multisig contract. Commit: 91c9cee976df029407030bbaec5a40093a64e9ab.

CSU-26 | INCONSISTENCY IN `Multisig`

Category	Severity	Location	Status
Inconsistency	Minor	contracts/Multisig.sol (Update3): 292, 327, 379	Resolved

Description

`executeTransaction()`, `removeTransaction()`, and `cancelTransaction()` can all be applied on transactions existing, valid, and not executed if `approvalCount[transactionId] >= required`.

This is an inconsistent design as a transaction with enough approval could be considered as cancellable/removable and executable at the same time.

Recommendation

We recommend removing the `approvalCount[transactionId] >= required` conditions from both `cancelTransaction()` and `removeTransaction()`. By definition, once a transaction has met the required approval threshold, it should be considered executable.

Alleviation

[CertiK, 05/22/2025]: The team addressed the issue by discontinuing their custom multisig implementation and opting to use a Safe multisig contract. Commit: 91c9cee976df029407030bbaec5a40093a64e9ab.

CSU-04 | INFORMATION NEEDED ON UPGRADE HANDLING

Category	Severity	Location	Status
Coding Issue	Informational		Resolved

Description

Contracts `Cngn2` and `Operation2` are upgradeable. Please clarify whether the source code provided is for an upgrade of an existing deployment, or whether this is the implementation code for a proxy being deployed for the first time. This information is needed in order to determine whether storage collisions during upgrades should be considered.

Recommendation

If the currently audited codebase is an upgrade to an existing deployment, we recommend providing the address for the currently existing contract logic in order to assess the potential for storage collisions. Otherwise, please confirm this is the contract logic to be used with the first deployment of the project.

Alleviation

[CertiK, 05/15/2025]:

Cngn

The upgrade from Cngn:0x4e1284b3fc4dc9b446788e07e2040c1b2705e713 to Cngn2.sol:addd17ef4aa2e812ec86f551532836811ab50dcc introduces no storage-slot collisions or layout mismatches.

Operations

The upgrade from Operations.sol:a3593071a40a058354753c6a0d7cb5c10a87a3e6 to Operations2.sol:addd17ef4aa2e812ec86f551532836811ab50dcc introduces no storage-slot collisions or layout mismatches.

CSU-14 | ARRAY MISSING `pop` FUNCTION

Category	Severity	Location	Status
Volatile Code	Informational	contracts/Multisig.sol (Base): 42	Resolved

Description

Arrays without the pop operation in Solidity can lead to inefficient memory management and increase the likelihood of out-of-gas errors.

```
42      Transaction[] public transactions; // Array of transactions
```

Recommendation

Consider adding functionality to remove elements from the array to prevent it from becoming too large over the lifetime of the contract.

Alleviation

[WrappedCBDC, 05/15/2025]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/wrappedcbdc/stablecoin-cngn/commit/f0a359deac8515fbd9ca5d2391a781bd57aa4324

[CertiK, 05/15/2025]: The team heeded the advice and resolved the issue in the commit hash: https://github.com/wrappedcbdc/stablecoin-cngn/commit/f0a359deac8515fbd9ca5d2391a781bd57aa4324

[CertiK, 05/22/2025]: The team addressed the issue by discontinuing their custom multisig implementation and opting to use a Safe multisig contract. Commit: 91c9cee976df029407030bbaec5a40093a64e9ab.

CSU-15 | COMPARISON TO BOOLEAN CONSTANT

Category	Severity	Location	Status
Coding Style	Informational	contracts/Cngn2.sol (Base): 246~278; contracts/Multisig.sol (Base): 30~33; contracts/Operations2.sol (Base): 93~101	Resolved

Description

Boolean constants can be used directly and do not need to be compared to true or false.

```
31          require(transactions[transactionId].executed == false,
"Transaction already executed");
```

```
266         require(
267             IAdmin(adminOperationsContract).removeCanMint(signer) == true,
268             "Failed to revoke minting authorization"
269         );
```

```
 97         require(canMint[_User] == true);
```

```
270         require(
271             IAdmin(adminOperationsContract).removeMintAmount(signer) == true,
272             "Failed to remove mint amount"
273         );
```

Recommendation

We recommend removing the equality to the boolean constant.

Alleviation

[CertiK, 05/12/2025]: The team heeded the advice and resolved the issue in commit 2189cfbd989c7e35f32b93d3eec641ae31e9a84e.

[CertiK, 05/22/2025]: The team addressed the issue by discontinuing their custom multisig implementation and opting to use a Safe multisig contract. Commit: 91c9cee976df029407030bbaec5a40093a64e9ab.

CSU-16 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	contracts/Cngn2.sol (Base): 92, 99	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

[CertiK, 05/15/2025]: The team heeded the advice and resolved the issue in commit 983f2827b52e4cfb18bd733463c58716d5359c41.

CSU-17 | MISSING ERROR MESSAGE

Category	Severity	Location	Status
Coding Style	Informational	contracts/Operations2.sol (Base): 97	Resolved

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise adding error message to the linked **require** statement.

Alleviation

[CertiK, 05/12/2025]: The team heeded the advice and resolved the issue in commit d023b1b19ca50700a0b496cd438b6c442bc1a9f1.

CSU-19 | COMMENTED CODE

Category	Severity	Location	Status
Coding Style	Informational	contracts/Multisig.sol (Base): 113~126	Resolved

Description

Commented code does not affect the functionality of the codebase and appears to be either remnants of test code or older functionality.

Recommendation

We recommend removing the redundant code to better prepare the code for production environments.

Alleviation

[CertiK, 05/12/2025]: The team heeded the advice and resolved the issue in commit 8a765f06776bb2141bc840e974101e17a2860c50.

[CertiK, 05/22/2025]: The team addressed the issue by discontinuing their custom multisig implementation and opting to use a Safe multisig contract. Commit: 91c9cee976df029407030bbaec5a40093a64e9ab.

CSU-24 | `executeTransaction()` IS SET AS `internal`

Category	Severity	Location	Status
Design Issue	Informational	Multisig.sol (Update2): 277~278	Resolved

Description

The changes introduced in a previous commit set `executeTransaction()` as an `internal` function. However, due to an update allowing a transaction to have more approval than required, the function should now be callable directly, so a transaction with enough approval can be executed without calling intermediary functions.

Recommendation

We recommend setting `executeTransaction()` as `public`.

Alleviation

[WrappedCBDC, 05/15/2025]: This was intentional to be internal. The approveTransaction calls the executeTransaction once all required signers have been met. Instead of calling the executeTransaction differently.

[CertiK, 05/15/2025]: The team acknowledged the issue and decided to remain unchanged in the scope of the audit.

[CertiK, 05/22/2025]: The team addressed the issue by discontinuing their custom multisig implementation and opting to use a Safe multisig contract. Commit: 91c9cee976df029407030bbaec5a40093a64e9ab.

OPTIMIZATIONS | CNGN STABLECOIN UPDATE

ID	Title	Category	Severity	Status
CSU-01	State Variable That Could Be Declared Immutable	Coding Issue	Optimization	Resolved
CSU-20	Inconsistency In `removeOwner()`	Code Optimization	Optimization	Resolved
CSU-25	Inconsistent Terminal Transaction Functions	Code Optimization	Optimization	Resolved

CSU-01 STATE VARIABLE THAT COULD BE DECLARED IMMUTABLE

Category	Severity	Location	Status
Coding Issue	Optimization	contracts/Multisig.sol (Base): 6	Resolved

Description

State variables that are not updated following deployment should be declared immutable to save gas.

Recommendation

We recommend adding the immutable attribute to state variables that never change after being set in the constructor.

Alleviation

[CertiK, 05/15/2025]: The team heeded the advice and resolved the issue in commit b5ffb7e69146dd33f6a1eae5ba7b9e9ff69da958.

[CertiK, 05/22/2025]: The team addressed the issue by discontinuing their custom multisig implementation and opting to use a Safe multisig contract. Commit: 91c9cee976df029407030bbaec5a40093a64e9ab.

CSU-20 | INCONSISTENCY IN `removeOwner()`

Category	Severity	Location	Status
Code Optimization	Optimization	Multisig.sol (Update1): 356~359, 375~377	Resolved

Description

The function `Multisig.removeOwner()` perfoms the following check:

```
356         require(
357             owners.length > required,
358             "Cannot have fewer owners than required signatures"
359         );
```

implying the function can execute normally only if `owners.length >= required + 1`, so only if `owners.length - 1 >= required`.

However, after removing the related owner, the following check is performed:

```
375         if (required > owners.length) {
376             changeRequirement(owners.length);
377         }
```

The statement will always be evaluated to `false`, so `changeRequirement(owners.length)` will never be executed and will be unreachable code.

Recommendation

We recommend removing the unreachable code.

Alleviation

[WrappedCBDC, 05/15/2025]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/wrappedcbdc/stablecoin-cngn/commit/f752865e8b38b391572782b40af0826786a29992

[CertiK, 05/15/2025]: The team heeded the advice and resolved the issue in the commit hash: https://github.com/wrappedcbdc/stablecoin-cngn/commit/f752865e8b38b391572782b40af0826786a29992

[CertiK, 05/22/2025]: The team addressed the issue by discontinuing their custom multisig implementation and opting to use a Safe multisig contract. Commit: 91c9cee976df029407030bbaec5a40093a64e9ab.

CSU-25 | INCONSISTENT TERMINAL TRANSACTION FUNCTIONS

Category	Severity	Location	Status
Code Optimization	Optimization	contracts/Multisig.sol (Update3): 314~315, 341~342, 366~367	Resolved

Description

The three “terminal” functions— `cancelTransaction()` , `markExpiredTransaction()` , and `removeTransaction()` — behave inconsistently, leading to confusion and on-chain bloat:

- **`cancelTransaction()`** Marks a valid, non-executed transaction as **inactive** without cleaning up storage.
- **`markExpiredTransaction()`** Marks a valid, non-executed transaction as **expired** without cleaning up storage.
- **`removeTransaction()`** Deletes a valid, non-executed transaction and clears all approval mappings (swap-and-pop).

These overlaps cause specific issues:

1. Calling `cancelTransaction()` on an old proposal prevents it ever being marked expired via `markExpiredTransaction()` .
2. Once a transaction is cancelled or expired, it **cannot** be removed through `removeTransaction()` , so stale entries linger.
3. A user can choose to cancel or remove a transaction under the same conditions, but the effects differ—creating unclear UX and maintenance headaches.

Recommendation

We recommend following the design:

- **`cancelTransaction()`**
  - Only the **creator** may call.
  - Applies only to valid, pending, non-expired transactions.
  - No storage cleanup
- **`markExpiredTransaction()`**
  - Callable by **anyone** once `block.timestamp > expirationTime` .

- Marks status as **Expired** (no storage cleanup).

- `removeTransaction()`
  - Callable by **anyone** but only on **Cancelled** or **Expired** transactions.
  - Performs swap-and-pop and clears all related mappings to reclaim storage.

Alleviation

[CertiK, 05/22/2025]: The team addressed the issue by discontinuing their custom multisig implementation and opting to use a Safe multisig contract. Commit: 91c9cee976df029407030bbaec5a40093a64e9ab.

FORMAL VERIFICATION | CNGN STABLECOIN UPDATE

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 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 (note that overflow properties were excluded from the verification):

Property Name	Title
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Transfers
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address
erc20-approve-never-return-false	`approve` Never Returns `false`
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-allowance-correct-value	`allowance` Returns Correct Value

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

Verification of Standard Ownable Properties

We verified *partial* properties of the public interfaces of those token contracts that implement the Ownable interface. This involves:

- function `owner` that returns the current owner,
- functions `renounceOwnership` that removes ownership,

- function `transferOwnership` that transfers the ownership to a new owner.

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

Property Name	Title
ownable-transferownership-correct	Ownership is Transferred
ownable-owner-succeed-normal	`owner` Always Succeeds
ownable-renounceownership-correct	Ownership is Removed
ownable-renounce-ownership-is-permanent	Once Renounced, Ownership Cannot be Regained

Verification Results

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 Cngn2 (contracts/Cngn2.sol) In Commit a3593071a40a058354753c6a0d7cb5c10a87a3e6

Verification of Pausable ERC-20 Compliance

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-correct-allowance	Inapplicable	The property does not apply to the contract
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-false	True	
erc20pausable-transferfrom-revert-paused	True	
erc20-transferfrom-revert-zero-argument	True	

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-exceed-balance	True	
erc20pausable-transfer-revert-paused	True	
erc20-transfer-never-return-false	True	
erc20-transfer-revert-zero	True	
erc20-transfer-false	True	
erc20-transfer-correct-amount	False	finding merged

Detailed Results for Function `approve`

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

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 `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 `totalSupply`

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

Detailed Results For Contract Admin2 (contracts/Operations2.sol) In Commit a3593071a40a058354753c6a0d7cb5c10a87a3e6

Verification of Standard Ownable Properties

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	True	

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	True	

Property Name	Final Result	Remarks
ownable-renounce-ownership-is-permanent	Inapplicable	The property does not apply to the contract

APPENDIX | CNGN STABLECOIN UPDATE

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.
Concurrency	Concurrency findings are about issues that cause unexpected or unsafe interleaving of code executions.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
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 ERC-20-Pausable Properties

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-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 `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 `Cngn2` 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-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 `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;
```

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 `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 `totalSupply`

erc20-totalsupply-change-state

The `totalSupply` function in contract Cngn2 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 Cngn2.

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;
```

Description of the Analyzed Ownable Properties

Properties related to function `transferOwnership`

ownable-transferownership-correct

Invocations of `transferOwnership(newOwner)` must transfer the ownership to the `newOwner`.

Specification:

```
ensures this.owner() == newOwner;
```

Properties related to function `owner`

ownable-owner-succeed-normal

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

Specification:

```
reverts_only_when false;
```

Properties related to function `renounceOwnership`

ownable-renounce-ownership-is-permanent

The contract must prohibit regaining of ownership once it has been renounced.

Specification:

```
constraint \old(owner()) == address(0) ==> owner() == address(0);
```

ownable-renounceownership-correct

Invocations of `renounceOwnership()` must set ownership to address(0).

Specification:

```
ensures this.owner() == address(0);
```

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.