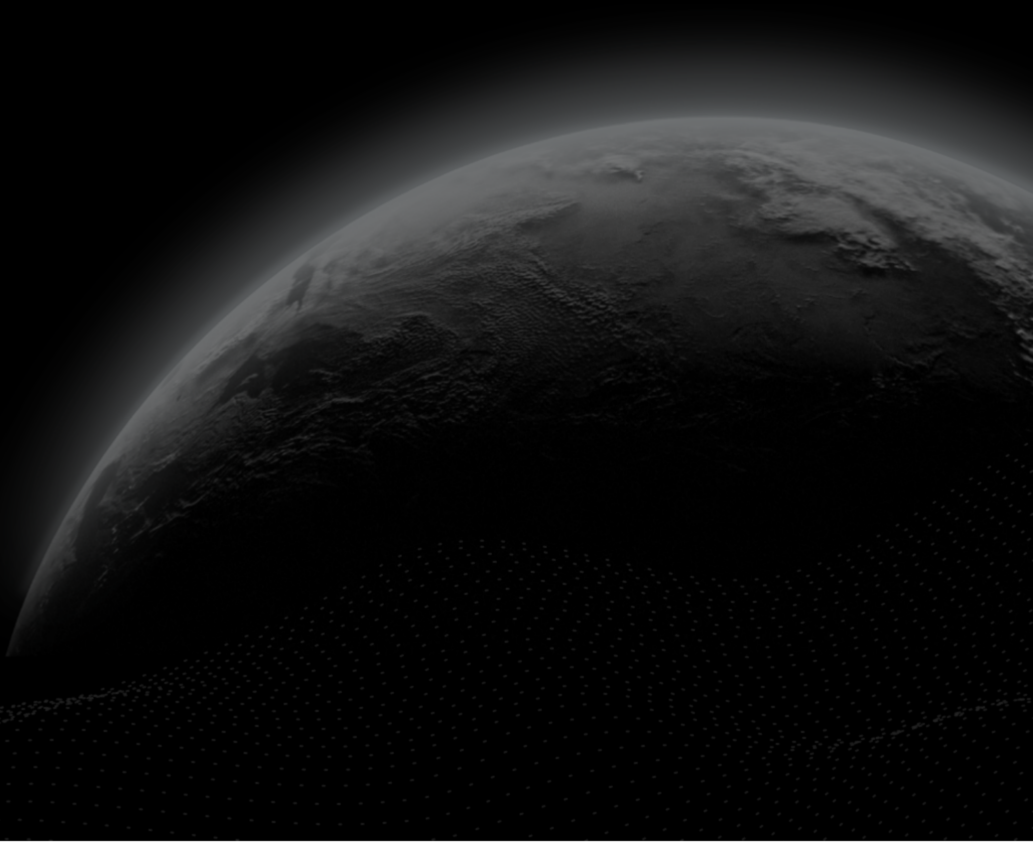
CERTIK

Preliminary Comments

Zchain - Audit 1

CertiK Assessed on Jul 12th, 2024



CertiK Assessed on Jul 12th, 2024

Zchain - Audit 1

These preliminary comments were prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Polygon (MATIC)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 07/12/2024	N/A

CODEBASE

- https://github.com/GoerBridge/GoerBridge-Smart-Contract/tree/main/contracts
- https://github.com/ZChain-168168/ZChain-staking-

View All in Codebase Page

COMMITS

- GoerBridge-Smart-Contract: 52e6094393ab40859e0581499eea54b356b3c154
- ZChain-staking-contracts:

View All in Codebase Page

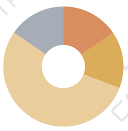
Highlighted Centralization Risks

ⓘ Privileged role can remove users' tokens

ⓘ Transfers can be paused

ⓘ Fees are bounded by 10%

Vulnerability Summary



13	0	0	0	0	0	13
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Pending

🟥 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.
🟧 2	Major	2 Pending	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
🟨 2	Medium	2 Pending	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨 7	Minor	7 Pending	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.

PRELIMINARY COMMENTS CERTIK

■ 0	Informational		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.
■ 2	Discussion	2 Pending	The impact of the issue is yet to be determined, hence requires further clarifications from the project team.

PRELIMINARY COMMENTS

TABLE OF CONTENTS | ZCHAIN - AUDIT 1

PRELIMINARY COMMENTS

PRELIMINARY COMMENTS

CODEBASE | ZCHAIN - AUDIT 1

Repository

- https://github.com/GoerBridge/GoerBridge-Smart-Contract/tree/main/contracts
- https://github.com/ZChain-168168/ZChain-staking-contracts/blob/master/contracts/Staking.sol

Commit

- GoerBridge-Smart-Contract: 52e6094393ab40859e0581499eea54b356b3c154
- ZChain-staking-contracts: 873152ef525196032990ad7f604a6893a37754ae

PRELIMINARY COMMENTS

AUDIT SCOPE | ZCHAIN - AUDIT 1

3 files audited ● 3 files without findings

ID	Repo	File	SHA256 Checksum
● SZC	ZChain-168168/ZChain-staking-contracts	Staking.sol	691261faf46b64801b74bbfb24f74aeaa4dc199a90236bdc8f5b4c20a4558cdd
● FGB	GoerBridge/GoerBridge-Smart-Contract	Factory.sol	0caaddc7913448f42275c7c4b90274caa96f4a6373d6e8918baf369caca1672d
● BGB	GoerBridge/GoerBridge-Smart-Contract	Bridge.sol	a922cc263e4a1c1743ed8e1c52d313ff751802487ddb3c28c290845b19ec2bb2

PRELIMINARY COMMENTS

APPROACH & METHODS | ZCHAIN - AUDIT 1

This report has been prepared for Zchain to discover issues and vulnerabilities in the source code of the Zchain - Audit 1 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.

PRELIMINARY COMMENTS

REVIEW NOTES | ZCHAIN - AUDIT 1

Overview

The **Zchain** project manages a suite of smart contracts intended for deployment on EVM platforms. The files under the audit scope include:

- The `Staking` contract, which enables users to become validators by staking Ether, provided they meet a predefined staking threshold. It manages validator onboarding, offboarding, and staked amounts through mappings and arrays, ensuring that only EOAs can interact with staking functions, and enforcing minimum and maximum validator counts.
- The `Factory` contract, which allows users to deploy new bridge instances by cloning an existing one, with a fee paid to a project wallet. It permits the owner to manage bridge origins and fees, and emits events upon deployment.
- The `Bridge` contract, which facilitates token transfers into this contract, charging fees. Key features include roles for the owner and monitor, EOA-only interaction, and pausable operations.

External Dependencies

In **Zchain**, 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 following values are initialized either at deploy time or by specific functions in smart contracts.

Factory:

- `bridgeClone` .

Bridge:

- `token` .

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 libraries/contracts are considered as third-party dependencies:

- @openzeppelin/contracts/

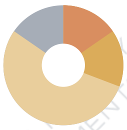
Privileged Functions

In the **Zchain** project, the privileged roles are adopted to ensure the dynamic runtime updates of the project, which are specified in the following finding: `Centralization Risks` .

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 | ZCHAIN - AUDIT 1



13	0	2	2	7	0	2
Total Findings	Critical	Major	Medium	Minor	Informational	Discussion

This report has been prepared to discover issues and vulnerabilities for Zchain - Audit 1. Through this audit, we have uncovered 13 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
BGB-02	**Withdrawal Centralization Risk**	**Centralization**	**Major**	● **Pending**
GBS-03	**Centralization Risks**	**Centralization**	**Major**	● **Pending**
SZC-04	Ineffective `isContract` Check	Volatile Code	Medium	● Pending
SZC-05	Lock-In Risk For Validators Due To Unstake Restrictions	Logical Issue	Medium	● Pending
BGB-03	Risk Of Losing Admin Access	Logical Issue	Minor	● Pending
BGB-04	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	● Pending
BGB-05	Missing Zero Address Validation	Volatile Code	Minor	● Pending
BGB-06	Incompatibility With Deflationary Tokens	Volatile Code	Minor	● Pending
BGB-07	Owner Able To Revoke Own Role	Inconsistency	Minor	● Pending
GBS-04	Usage Of `transfer` / `send` For Sending Native Tokens	Volatile Code	Minor	● Pending
GBS-05	The Surplus Native Tokens Are Not Returned	Design Issue	Minor	● Pending

ID	Title	Category	Severity	Status
BGB-01	Discussion On `bridgeFee` Collection In `receiveTokens` Function	Design Issue	Discussion	● Pending
FGB-02	Discussion On Unchecked `projectId`	Design Issue	Discussion	● Pending

BGB-02 | WITHDRAWAL CENTRALIZATION RISK

Category	Severity	Location	Status
Centralization	● **Major**	**Bridge.sol (Bridge): 180~181**	● **Pending**

Description

This contract features a `withdrawToken` function that allows the `_owner` role to withdraw the entire token balance from the contract without any restriction, including tokens meant to be locked by users. Should this account become compromised, it would result in the loss of all user tokens, leading to substantial financial losses for users as wrapped tokens will no longer be backed by locked tokens:

```
function withdrawToken(uint256 amount, address payable receiverWallet) external
onlyOwner returns (bool) {
    ...
}
```

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.

GBS-03 | CENTRALIZATION RISKS

Category	Severity	Location	Status
Centralization	● **Major**	**Bridge.sol (Bridge): 61, 98, 180, 193, 198, 208, 219, 225, 243, 250, 267, 307, 320, 335, 357, 378, 382; Factory.sol (Factory): 73~74, 80~81**	● **Pending**

Description

In the `Factory` contract, the `_owner` has authority over the following functions:

- `setListBridge` : Sets the status of a bridge contract in the `listBridge` mapping.
- `setDeployFee` : Updates the fee required for deploying a new bridge contract.

This contract extends from `Ownale` , the `_owner` has authority over the following functions:

- `renounceOwnership` : Leaves the contract without owner.
- `transferOwnership` : Transfers ownership of the contract to a new account (`newOwner`).

In the `Bridge` contract,

- the `DEFAULT_ADMIN_ROLE` has authority over the following functions:
    - `withdrawToken` : Withdraws tokens to a specified wallet.
    - `setFeeNative` : Sets the native fee for token transfers.
    - `addMonitor` : Assigns a monitor role to an account.
    - `delMonitor` : Revokes the monitor role from an account.
    - `transferOwnership` : Transfers the contract ownership to a new owner.
    - `setMinTokenAmount` : Defines the minimum token amount required for a blockchain.
    - `setFeePercentageBridge` : Adjusts the fee percentage for the bridge.
    - `addBlockchainFrom` : Adds a new blockchain to the `blockchainInfoFrom` list.
    - `addBlockchainTo` : Registers a new blockchain to the transferring list with a minimum token amount.
    - `delBlockchainFrom` : Removes a blockchain from the `blockchainInfoFrom` list.
    - `delBlockchainTo` : Deletes a blockchain from the transferring list.
    - `pause` : Halts all contract functions.
    - `unpause` : Resumes all contract functions.
    - `renounceRole` : Allows an account to renounce a role it possesses.
    - `revokeRole` : Revokes a previously granted role from an account.

- the `MONITOR_ROLE` role has authority over the following functions:
    - `acceptTransfer` : Validates and processes a token transfer from another blockchain to a specified receiver address.

Any compromise of the privileged accounts may allow a hacker to take advantage of this authority to modify critical state variables, withdraw fees from the project, pause normal activities, and significantly impact the project's functionalities.

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:

PRELIMINARY COMMENTS CERTIK

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.

PRELIMINARY COMMENTS

SZC-04 | INEFFECTIVE `isContract` CHECK

Category	Severity	Location	Status
Volatile Code	● Medium	Staking.sol (Staking): 32	● Pending

Description

The `isContract` check cannot cover all scenarios. The check can be bypassed if the call is from the constructor of a contract or when the contract is destroyed. In that case, `isContract` will return `false` because its `codesize` will be zero.

The comment in the `isContract` implementation from the OpenZeppelin library also says that "it is unsafe to assume that an address for which this function returns false is an externally-owned account (EOA) and not a contract."

- Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/3dac7bbed7b4c0dbf504180c33e8ed8e350b93eb/contracts/utils/Address.sol

```
        require(!msg.sender.isContract(), "Only EOA can call function");
```

Recommendation

We recommend adding an additional `<address> == tx.origin` check to cover all the scenarios.

For example,

```
modifier notContract() {
    require(!(msg.sender.isContract()) && (msg.sender == tx.origin), "Only EOA can call function");
    _;
}
```

Please note that while the check works for the current EVM version, future updates to the EVM (ex. EIP-3074) might cause the check to become ineffective.

SZC-05 | LOCK-IN RISK FOR VALIDATORS DUE TO UNSTAKE RESTRICTIONS

Category	Severity	Location	Status
Logical Issue	● Medium	Staking.sol (Staking): 124~125, 131~132, 140~141	● Pending

Description

The `Stake` contract's `unstake` function is designed with a restriction that prevents validators from withdrawing their staked Ether if the total number of validators is at or below the `_minimumNumValidators` threshold. The conditional check:

```
require(
    _validators.length > _minimumNumValidators,
    "Validators can't be less than the minimum required validator num"
);
```

prevents some validators from unstaking due to a low number of validators, creating a situation where there are at most `_minimumNumValidators` validators with funds locked in the contract.

This design flaw not only hinders validators from unstaking but also discourages new users from becoming validators due to the fear of their assets being locked.

Furthermore, if the system is meant to be a Proof-of-Stake mechanism where stakers are able to acquire rewards, the system can be gamed by splitting stakes into multiple externally owned accounts (EOAs), each depositing below the validator threshold. These EOAs are not recognized as validators and thus can unstake without restriction, circumventing the intended security measure.

Recommendation

The unstake function should be modified to allow validators to withdraw their funds regardless of the current number of active validators, perhaps by introducing a time-based lockup period or a gradual unstaking mechanism that ensures network security without indefinitely locking user funds.

Additionally, consider implementing measures to prevent gaming the system through the use of multiple EOAs to bypass the validator threshold.

PRELIMINARY COMMENTS

BGB-03 | RISK OF LOSING ADMIN ACCESS

Category	Severity	Location	Status
Logical Issue	● Minor	Bridge.sol (Bridge): 243~244	● Pending

Description

If the address to be set as the new admin role is the same as the current admin role's address, it will result in the admin role losing its administration authorization over the contract.

This vulnerability arises because the function first grants the admin role to the new owner's address and then revokes the admin role from the current admin's address(i.e., `msg.sender`). However, if the new owner's address matches the current one, the admin role will be revoked prematurely, leading to the admin losing the authorization for contract administration.

Recommendation

We recommend adding validation to ensure that the address to be set as the admin role is different from the current admin role's address.

PRELIMINARY COMMENTS

BGB-04 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	● Minor	Bridge.sol (Bridge): 86, 87, 168, 183	● Pending

Description

The return values of the `transfer()` and `transferFrom()` calls in the smart contract are not checked. Some ERC-20 tokens' transfer functions return no values, while others return a bool value, they should be handled with care. If a function returns `false` instead of reverting upon failure, an unchecked failed transfer could be mistakenly considered successful in the contract.

```
86          IERC20(token).transferFrom(_msgSender(), address(this), amountMinusFees
);
```

```
87          IERC20(token).transferFrom(_msgSender(), OWNER_WALLET, bridgeFee);
```

```
168         IERC20(token).transfer(to, amount);
```

```
183         IERC20(token).transfer(receiverWallet, amount);
```

Recommendation

It is advised to use the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if false is returned, making it compatible with all ERC-20 token implementations.

PRELIMINARY COMMENTS CERTIK

BGB-05 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	Bridge.sol (Bridge): 124, 188, 244	● Pending

Description

Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities. For example, transferring tokens to a zero address can result in a permanent loss of those tokens.

```
124        OWNER_WALLET = owner;
```

- `owner` is not zero-checked before being used.

```
188          receiverWallet.transfer(amount);
```

- `receiverWallet` is not zero-checked before being used.

```
244        OWNER_WALLET = newOwner;
```

- `newOwner` is not zero-checked before being used.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

PRELIMINARY COMMENTS

BGB-06 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS

Category	Severity	Location	Status
Volatile Code	● Minor	Bridge.sol (Bridge): 86~87, 87~88	● Pending

Description

The project design may not be compatible with non-standard ERC20 tokens, such as deflationary tokens or rebase tokens.

The functions use `transferFrom()` to move funds from the sender to the recipient but fail to verify if the received token amount matches the transferred amount. This could pose an issue with fee-on-transfer tokens, where the post-transfer balance might be less than anticipated, leading to balance inconsistencies.

Scenario

When transferring deflationary ERC20 tokens, the input amount may not equal the received amount due to the charged transaction fee. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrive to the contract. However, a failure to discount such fees may allow 100 tokens to be transferred to the recipient blockchain, which causes the project to lose 10 tokens in such a transaction.

Recommendation

We advise the client to regulate the set of tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support non-standard ERC20 tokens.

BGB-07 | OWNER ABLE TO REVOKE OWN ROLE

Category	Severity	Location	Status
Inconsistency	● Minor	Bridge.sol (Bridge): 231	● Pending

Description

The owner, which is the address with the `DEFAULT_ADMIN_ROLE` role, should not be allowed to remove their role, as shown in the override to `renounceRole()` :

```
219    function renounceRole(bytes32 role, address account) public virtual override
{
220      require(role != DEFAULT_ADMIN_ROLE, "can not renounce role owner");
221      require(account == _msgSender(), "can only renounce roles for self");
222      super.renounceRole(role, account);
223    }
```

However, the `revokeRole()` function does not have the same restriction:

```
225    function revokeRole(bytes32 role, address account)
226      public
227      virtual
228      override
229      onlyRole(getRoleAdmin(role))
230    {
231      super.revokeRole(role, account);
232    }
```

Since the role admin for the `DEFAULT_ADMIN_ROLE` role is itself, the owner is able to renounce their role through `revokeRole()` .

Recommendation

It is recommended to not allow the owner to renounce their role through `revokeRole()` .

CERTIK PRELIMINARY COMMENTS

GBS-04 | USAGE OF `transfer` / `send` FOR SENDING NATIVE TOKENS

Category	Severity	Location	Status
Volatile Code	● Minor	Bridge.sol (Bridge): 84, 89~90, 165~166, 188~189; Factory.sol (Factory): 57	● Pending

Description

It is not recommended to use Solidity's `transfer()` and `send()` functions for transferring native tokens, since some contracts may not be able to receive the funds. Those functions forward only a fixed amount of gas (2300 specifically) and the receiving contracts may run out of gas before finishing the transfer. Also, EVM instructions' gas costs may increase in the future. Thus, some contracts that can receive now may stop working in the future due to the gas limitation.

```
84              payable(OWNER_WALLET).transfer(msg.value);
```

```
89              payable(OWNER_WALLET).transfer(msg.value);
```

- `Bridge.receiveTokens` uses `transfer()`.

```
165             to.transfer(amount);
```

- `Bridge._sendToken` uses `transfer()`.

```
188             receiverWallet.transfer(amount);
```

- `Bridge.withdrawToken` uses `transfer()`.

```
57          payable(projectWallet).transfer(deployFee);
```

- `Factory.deployedBridge` uses `transfer()`.

Recommendation

PRELIMINARY COMMENTS

We recommend using the `Address.sendValue()` function from OpenZeppelin.

Since `Address.sendValue()` may allow reentrancy, we also recommend guarding against reentrancy attacks by utilizing the Checks-Effects-Interactions Pattern or applying OpenZeppelin ReentrancyGuard.

PRELIMINARY COMMENTS

GBS-05 | THE SURPLUS NATIVE TOKENS ARE NOT RETURNED

Category	Severity	Location	Status
Design Issue	● Minor	Bridge.sol (Bridge): 84~85; Factory.sol (Factory): 56~57	● Pending

Description

In the `deployedBridge` function, there's a validation to ensure `msg.value` meets the minimum required amount of native tokens. However, it's important to note that if `msg.value` exceeds this amount, the function currently lacks a mechanism to refund the excess. This could lead to unintentional loss of funds for the caller, as any surplus in `msg.value` is not returned. Implementing a refund logic for the excess amount is crucial to prevent the potential loss of caller funds.

The same issue also occurs in the `receiveTokens` function.

Recommendation

We recommend adding logic for refunding surplus or modifying the validation process to enforce that the amount of native tokens paid by the caller exactly matches the required amount.

CERTIK PRELIMINARY COMMENTS

BGB-01 DISCUSSION ON `bridgeFee` COLLECTION IN `receiveTokens` FUNCTION

Category	Severity	Location	Status
Design Issue	● Discussion	Bridge.sol (Bridge): 90~91	● Pending

Description

The `receiveTokens` function allows anyone to send either native tokens or specified tokens to this contract, according to the following code snippets:

```
if(token != ZERO_ADDRESS) {
    if(FEE_NATIVE > 0) {
        payable(OWNER_WALLET).transfer(msg.value);
    }
    IERC20(token).transferFrom(_msgSender(), address(this), amountMinusFees);
    IERC20(token).transferFrom(_msgSender(), OWNER_WALLET, bridgeFee);
} else {
    payable(OWNER_WALLET).transfer(FEE_NATIVE);
    payable(address(this)).transfer(amountMinusFees);
}
```

If the `token` state variable is not the zero address, the user's transferred value consists of two parts: `msg.value` and `amount`. In this case:

- `msg.value` is transferred to the `OWNER_WALLET`.
- The `amount` is split into two parts:
    - `bridgeFee` is transferred from `msg.sender` to the `OWNER_WALLET`.
    - The remaining `amountMinusFees` is kept in this contract.

If the `token` is the zero address:

- Only the `FEE_NATIVE` is transferred to the `OWNER_WALLET`.
- The remaining value stays in this contract, and `bridgeFee` is not transferred to the `OWNER_WALLET`.

We need to discuss this design with the team to ensure it is intentional and correct. Additionally, the line `payable(address(this)).transfer(amountMinusFees)` is unnecessary, as the contract already holds the native tokens sent by the user in `msg.value`.

FGB-02 | DISCUSSION ON UNCHECKED `projectId`

Category	Severity	Location	Status
Design Issue	● Discussion	Factory.sol (Factory): 48~49	● Pending

Description

The `deployedBridge` function allows anyone to create a new bridge instance by using an existing one and paying the `deployFee` to the `projectWallet` . The function includes an argument `projectId` , which is specified by the caller. The function does not validate this value; it simply sets `checkUnique[contractOrigin][projectId]` to true regardless of the original value of this mapping. Note that `projectId` is only used in the event and does not affect the creation of the bridge contract.

```
function deployedBridge(
        address contractOrigin,
        address token,
        address monitorAccount,
        string memory projectId,
        uint256 feeNative,
        uint256 feePercentBridge
    ) public payable {
        ...
        IBridge(bridgeClone).setDeployBridge(token, _msgSender(), monitorAccount,
feeNative, feePercentBridge);
        checkUnique[contractOrigin][projectId] = true;

        emit DeployedBridge(
            projectId,
            bridgeClone,
            _msgSender(),
            monitorAccount,
            contractOrigin,
            token,
            deployFee
        );
```

We would like to discuss this design with the team to ensure it is intentional and correct.

PRELIMINARY COMMENTS

OPTIMIZATIONS | ZCHAIN - AUDIT 1

ID	Title	Category	Severity	Status
FGB-01	State Variable Should Be Declared Constant	Coding Issue	Optimization	● Pending
GBS-01	Inefficient Memory Parameter	Inconsistency	Optimization	● Pending
GBS-02	Unnecessary Use Of SafeMath	Coding Issue	Optimization	● Pending
SZC-01	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	● Pending
SZC-02	User-Defined Getters	Gas Optimization	Optimization	● Pending

PRELIMINARY COMMENTS

FGB-01 | STATE VARIABLE SHOULD BE DECLARED CONSTANT

Category	Severity	Location	Status
Coding Issue	● Optimization	Factory.sol (Factory): 29, 30	● Pending

Description

State variables that never change should be declared as `constant` to save gas.

```
29      string private _name = "Bridge Factory V1";
```

- `_name` should be declared `constant`.

```
30      address payable public projectWallet = payable(
0x5412121507C1aBaEBE39271c34Ea7dD10eba22D8); // GoerBridge Deployer
```

- `projectWallet` should be declared `constant`.

Recommendation

We recommend adding the `constant` attribute to state variables that never change.

PRELIMINARY COMMENTS

GBS-01 | INEFFICIENT MEMORY PARAMETER

Category	Severity	Location	Status
Inconsistency	● Optimization	Bridge.sol (Bridge): 250; Factory.sol (Factory): 48	● Pending

Description

One or more parameters with `memory` data location are never modified in their functions and those functions are never called internally within the contract. Thus, their data location can be changed to `calldata` to avoid the gas consumption copying from calldata to memory.

```
function setMinTokenAmount(string memory blockchainName, uint256 newAmount)
```

`setMinTokenAmount` has memory location parameters: `blockchainName` .

```
    function deployedBridge(
        address contractOrigin,
        address token,
        address monitorAccount,
        string memory projectId,
        uint256 feeNative,
        uint256 feePercentBridge
    ) public payable {
```

`deployedBridge` has memory location parameters: `projectId` .

Recommendation

We recommend changing the parameter's data location to `calldata` to save gas.

- For Solidity versions prior to 0.6.9, since public functions are not allowed to have calldata parameters, the function visibility also needs to be changed to `external` .
- For Solidity versions prior to 0.5.0, since parameter data location is implicit, changing the function visibility to `external` will change the parameter's data location to calldata as well.

CERTIK PRELIMINARY COMMENTS

GBS-02 | UNNECESSARY USE OF SAFEMATH

Category	Severity	Location	Status
Coding Issue	● Optimization	Factory.sol (Factory): 15; SafeMath.sol (52e6094): 15	● Pending

Description

The `SafeMath` library is used unnecessarily. With Solidity compiler versions 0.8.0 or newer, arithmetic operations will automatically revert in case of integer overflow or underflow.

```
    using SafeMath for uint256;
```

- `SafeMath` library is used for `uint256` type in `Factory` contract.

```
library SafeMath {
```

- An implementation of `SafeMath` library is found.

Recommendation

We advise removing the usage of `SafeMath` library and using the built-in arithmetic operations provided by the Solidity programming language.

PRELIMINARY COMMENTS

SZC-01 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	● Optimization	Staking.sol (Staking): 18, 19	● Pending

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

PRELIMINARY COMMENTS

SZC-02 | USER-DEFINED GETTERS

Category	Severity	Location	Status
Gas Optimization	● Optimization	Staking.sol (Staking): 59~61, 77~79, 81~83, 85~87, 89~91	● Pending

Description

In the `Staking` contract, state variables declared as `public` automatically have a getter function. This contract contains explicit public view functions that duplicate the functionality of these auto-generated getters.

Recommendation

It is recommended to remove the manually written public view functions for state variables already declared as `public`.

CERTIK PRELIMINARY COMMENTS

FORMAL VERIFICATION | ZCHAIN - AUDIT 1

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-hasrole-succeed-always	`hasRole` Function Always Succeeds
accesscontrol-getroleadmin-succeed-always	`getRoleAdmin` Function Always Succeeds
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-change-state	`getRoleAdmin` Function Does Not Change State
accesscontrol-grantrole-correct-role-granting	`grantRole` Correctly Grants Role
accesscontrol-hasrole-change-state	`hasRole` Function Does Not Change State
accesscontrol-revokerole-correct-role-revoking	`revokeRole` Correctly Revokes Role

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

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 Bridge (contracts/Bridge.sol) In Commit 52e6094393ab40859e0581499eea54b356b3c154

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	

PRELIMINARY COMMENTS CERTIK

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 Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● 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 Factory (contracts/Factory.sol) In Commit 52e6094393ab40859e0581499eea54b356b3c154

Verification of Standard Ownable Properties

Detailed Results for Function `owner`

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

PRELIMINARY COMMENTS

Detailed Results for Function `renounceOwnership`

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

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	● True	

PRELIMINARY COMMENTS

APPENDIX | ZCHAIN - AUDIT 1

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
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.

CERTIK PRELIMINARY COMMENTS

- 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 `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 `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 `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:

CERTIK PRELIMINARY COMMENTS

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

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 Ownable Properties

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

PRELIMINARY COMMENTS

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

Specification:

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

Properties related to function `transferOwnership`

ownable-transferownership-correct

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

Specification:

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

PRELIMINARY COMMENTS

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

PRELIMINARY COMMENTS

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.

CERTIK PRELIMINARY COMMENTS

CertiK | **Securing** the **Web3** World

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.