SMART CONTRACT AUDIT

ZOKYO.

October 13th, 2021 | v. 1.0

PASS

Zokyo's Security Team has concluded that these smart contracts pass security qualifications and are fully production-ready

TECHNICAL SUMMARY

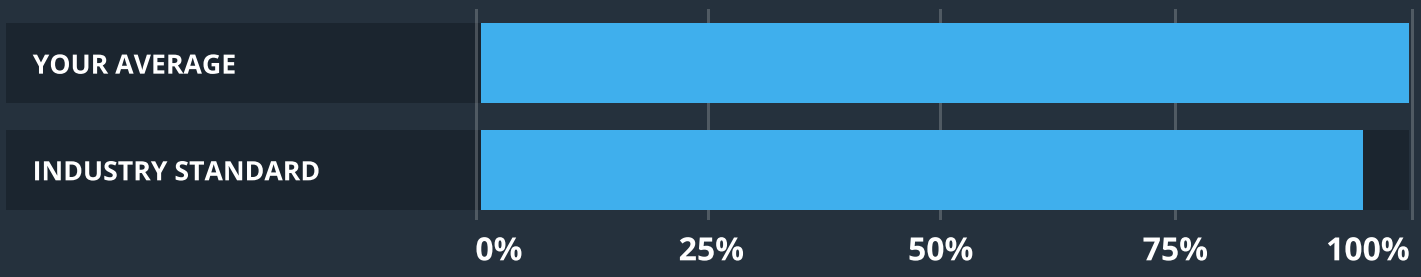
This document outlines the overall security of the Gamestation smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Gamestation smart contract codebase for quality, security, and correctness.

Contract Status

There were no critical issues found during the audit.

Testable Code



The testable code is 100%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Gamestation team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Gamestation repository.

Repository:
https://bitbucket.org/applicature/tokengear.contracts/src/gamestation-audit-20-08-2021/

Last commit:
fed31e9bc3d5fad5f7b4384373a32a886b557c4b

Contracts under the scope:
GamestationBridge

Throughout the review process, care was taken to ensure that the token contract:

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1	Due diligence in assessing the overall code quality of the codebase.
2	Cross-comparison with other, similar smart contracts by industry leaders.
3	Testing contract logic against common and uncommon attack vectors.
4	Thorough, manual review of the codebase, line-by-line.

SUMMARY

Zokyo security team has conducted a smart contract audit of the given codebase. During the process, we have found several issues. Among them were findings with high, medium, and low severity levels. No critical issues were identified.

The Gamestation team has taken into consideration all recommendations and successfully fixed the issues found. Hence, the contract bears no secure or operational risk to the contract owner or the end-user.

Based on the provided codebase and the outcome of this audit, we can state that the contracts are fully production-ready.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the ability of the contract to compile or operate in a significant way.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

HIGH | RESOLVED

The owner will be able to remove the token in the case when the liquidity is not zero and when the owner will try to call the adminWithdraw() for the deleted token it would be impossible to withdraw that token from the contract balance.

Re-audit
The issue related to removing the supported tokens was fixed in the function removeSupportedToken(). The function adminWithdraw() was deleted.

MEDIUM | RESOLVED

In the function addSupportedToken() there is no check if the added token is already on the list. There is a possibility to call the function with the same address of the token and rewrite the info about the current token.

MEDIUM | RESOLVED

Solidity filei has no license declaration.

Recommendation:
Specify license in every Solidity file.

LOW | RESOLVED

In the constructor, there is no check if the length of the array's tokensToSupport_ tokensInfo_ is the same. It can be the reason for the incorrect behavior.

LOW | RESOLVED

In the function withdraw(), addLiquidity(), adminWithdraw() there is no check if the requesting token is supported.

Recommendation:
Add checking if the token is supporting:
require(supportedTokens.contains(tokenAddress_), “Token is not supported”)

	GamestationBridge
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Gamestation team

Code Coverage

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
contracts\	95.71	89.40	94.58	97.49	
FixedSwap.sol	100.00	100.00	100.00	100.00	
GameStationBridge.sol	99.07	98.33	100.00	99.04	294
Vesting.sol	97.64	84.29	95.00	97.60	117, 354, 403
VestingFactory.sol	93.33	75.00	83.33	93.33	20
contracts\interfaces\	100.00	100.00	100.00	100.00	
IERC20Mintable.sol	100.00	100.00	100.00	100.00	
IFeeDistributor.sol	100.00	100.00	100.00	100.00	
IGameStationBridge.sol	100.00	100.00	100.00	100.00	
contracts\mock\	100.00	100.00	100.00	100.00	
MockFeeDistributor.sol	100.00	100.00	100.00	100.00	
contracts\tokens\	86.36	75.00	83.33	88.46	
GameStationToken.sol	100.00	100.00	100.00	100.00	
wGameStationToken.sol	72.73	50.00	66.67	76.92	26, 30, 42
contracts\utils\	22.08	23.08	22.45	26.47	
AttoDecimal.sol	15.79	14.29	17.07	18.37	... 184, 185, 197
TwoStageOwnable.sol	40.00	33.33	50.00	47.37	... 41, 42, 43, 44
All files	**81.19**	**77.49**	**80.07**	**82.48**	

Test Results

Contract: FixedSwap

Functions

createSimplePool

- ✓ should create simple pool correctly
- ✓ createSimplePool should fail if set wrong ending timestamp (790ms)
- ✓ createSimplePool should fail if set fee which higher than 100 (233ms)

createIntervalPool

- ✓ should create interval pool correctly
- ✓ should fail creating interval pool if immediately unlocking part value is not valid (1046ms)
- ✓ should fail creating interval pool if interval unlocking part is not valid (1010ms)
- ✓ should fail creating interval pool if interval starting timestamp is less than last interval starting timestamp (1075ms)
- ✓ should fail creating interval pool if last unlocking part is not equal to one (1287ms)
- ✓ should create interval pool correctly if startsAt will be bigger than timestamp (841ms)
- ✓ should fail creating interval pool if ending timestamp is less than starting (212ms)
- ✓ should fail creating interval pool if fee is equal 100% (248ms)

createLinearPool

- ✓ should create linear pool correctly
- ✓ should fail creating linear pool if linear unlocking less than or equal to pool ending timestamp (254ms)

poolProps

- ✓ should return pool properties (134ms)
- ✓ should fail return pool properties if pool does not exist (212ms)

intervalPoolProps

- ✓ should return interval pool properties (114ms)
- ✓ should fail return interval pool properties if pool does not exist (271ms)
- ✓ should fail return interval pool properties if pool is not interval (108ms)

linearPoolProps

- ✓ should return linear pool properties (92ms)
- ✓ should fail return linear pool properties if pool does not exist (200ms)
- ✓ should fail return linear pool properties if pool is not linear (109ms)

poolState

- ✓ should return pool state (100ms)

poolAccount

- ✓ should return pool type and account state (108ms)

intervalPoolAccount

✓ should return account state, complex account state and count of unlocked intervals (113ms)

linearPoolAccount

✓ should return account state, complex account state and immediately unlocked amount (127ms)

collectedFees

✓ should return collected fees of the token (274ms)

increaseIssuance

✓ should increase the pool token issuance by amount correctly (279ms)
✓ should fail increasing the pool token issuance if amount is zero (212ms)
✓ should fail increasing the pool token issuance if amount exceeds issuance limit (234ms)
✓ should fail increasing the pool token issuance if amount exceeds issuance limit (772ms)
✓ should fail increasing the pool token issuance if caller is not the owner (227ms)

createPaymentLimit

✓ should create new payment limit correctly (251ms)

changeLimit

✓ should change payment limit correctly (207ms)
✓ should fail changing payment limit if limit index does not exist (181ms)

setAccountsLimit

✓ should set limit for accounts correctly (465ms)
✓ should setting limit for accounts if accounts are not provided (215ms)
✓ should set limit for accounts if accounts already have limit with this index (223ms)

swap

✓ should make swap with payment amount and issuance amount (975ms)
✓ should fail swap if requested payment amount is zero (255ms)
✓ should fail swap if pool is not started yet (1076ms)
✓ should fail swap if pool is already ended (337ms)
✓ should fail swap if pool does not have available issuance (235ms)
✓ should fail swap if payment sum is bigger than payment limit (569ms)
✓ should make swap correctly if payment sum and payment amount will be bigger than payment limit (1964ms)
✓ should make swap correctly if issuance amount will be less than available amount (1145ms)
✓ should make swap correctly if calculated issuance amount equals zero (2003ms)
✓ should make swap correctly if calculated payment amount equals zero (1802ms)
✓ should make swap correctly if pool type is SIMPLE (1973ms)
✓ should make swap correctly if pool type is LINEAR (1007ms)
✓ should make swap correctly if issuance to withdraw will not be equal zero (2334ms)

unlockInterval

✓ should unlock interval correctly (368ms)
✓ should unlock interval correctly if issuance to Withdraw is not zero (1885ms)

```
    ✓ should fail unlocking interval if interval index does not exist (304ms)
    ✓ should fail unlocking interval if interval has not started yet (220ms)
    ✓ should fail unlocking interval if interval is already unlocked (556ms)
  unlockLinear
    ✓ should unlock linear correctly (1697ms)
    ✓ should unlock linear correctly if withdrawal amount equals zero (1767ms)
    ✓ should fail unlocking linear if pool is not ended (259ms)
    ✓ should fail unlocking linear if withdrawn issuance amount is less than issuance amount (394ms)
  withdrawPayments
    ✓ withdrawPayments should be failed if anyone hasn't sent any payments yet (282ms)
    ✓ withdrawPayments should be failed if caller isn't owner (261ms)
    ✓ withdrawPayments should widthdraw payments correctly (1796ms)
  withdrawUnsold
    ✓ withdrawUnsold should be failed if smartcontract hasn't ended yet (239ms)
    ✓ withdrawUnsold should be failed if caller isn't owner (215ms)
    ✓ withdrawUnsold should be failed if request wrong pool index (301ms)
    ✓ withdrawUnsold should be failed if unsold payments don't exist (340ms)
    ✓ withdrawUnsold should be withdraw correctly (806ms)
  collectFee
    ✓ collectFee should be failed if request wrong pool index (171ms)
    ✓ collectFee should be collect fee correctly (2526ms)
  withdrawFee
    ✓ withdrawFee should be failed if smart contract doesn't have collected fees (286ms)
    ✓ withdrawFee should be failed if caller ins't owner (295ms)
    ✓ withdrawFee should be withdraw fee correctly (3409ms)
  nominateNewPoolOwner
    ✓ nominateNewPoolOwner should be failed if caller isn't owner (354ms)
    ✓ nominateNewPoolOwner should be failed if owner tried to nominate himself (196ms)
    ✓ nominateNewPoolOwner should be nominate new owner correctly (302ms)
    ✓ nominateNewPoolOwner should be nominate new owner two times correctly (667ms)
  acceptPoolOwnership
    ✓ acceptPoolOwnership should be failed if caller wasn't nominated like a new owner (237ms)
    ✓ acceptPoolOwnership should be failed if request wrong pool index (279ms)
    ✓ acceptPoolOwnership should be accept correctly (1267ms)
Specifical Test Cases
  ✓ should swap correctly if user is not added (3141ms)
```

Contract: GGToken

- ✓ has a name (252ms)
- ✓ has a symbol (111ms)
- ✓ has 18 decimals (149ms)

pausable token

- ✓ fail when transfer, burn when pause (2644ms)
- ✓ fail if call not owner (941ms)
- ✓ fail if burning if allowance is less than amount (368ms)

total supply

- ✓ returns the total amount of tokens (216ms)

balanceOf

when the requested account has no tokens

- ✓ returns zero (115ms)

when the requested account has some tokens

- ✓ returns the total amount of tokens (84ms)

transfer

when the recipient is not the zero address

when the sender does not have enough balance

- ✓ reverts (901ms)

when the sender transfers all balance

- ✓ transfers the requested amount (920ms)
- ✓ emits a transfer event (219ms)

when the sender transfers zero tokens

- ✓ transfers the requested amount (1428ms)
- ✓ emits a transfer event (563ms)

when the recipient is the zero address

- ✓ reverts (419ms)

transfer from

when the token owner is not the zero address

when the recipient is not the zero address

when the spender has enough approved balance

when the token owner has enough balance

- ✓ transfers the requested amount (1745ms)
- ✓ decreases the spender allowance (127ms)

when the token owner does not have enough balance

- ✓ reverts (403ms)

when the spender does not have enough approved balance

when the token owner does not have enough balance

✓ reverts (918ms)
when the recipient is the zero address
✓ reverts (531ms)
when the token owner is the zero address
✓ reverts (385ms)

Contract: GameStationBridge
Check correct initialized
✓ should be fail if signers array is empty (1021ms)
✓ should be fail if arrays have different lengths (458ms)
✓ should be fail if fee is incorrect (531ms)
✓ should be faile if incorrect liquidity is not zero (625ms)
✓ Should correct initialized tokens types (193ms)
✓ Should correct initialized tokens support address (138ms)
IS_TOKEN_SUPPORTED
✓ Should return false if not support (144ms)
✓ Should return true if support (161ms)
getSupportedTokens
✓ Should return correct list (176ms)
✓ Should return empty after all delete (907ms)
✓ Should return correct list after add (1183ms)
addSigners
✓ Should fail if caller not owner (210ms)
✓ Should add correct signer (441ms)
getSignersAddress
✓ Should fail if caller not owner (91ms)
✓ Should get correct list (113ms)
removeSigners
✓ Should fail if caller not owner (221ms)
✓ Should fail if signer for delete not found (265ms)
✓ Should fail if signers array after delete is empty (587ms)
✓ Should correct delete signer (430ms)
setFeeDistributor
✓ Should correct set (318ms)
✓ Should fail if caller not owner (372ms)
setFeeRecipient
✓ Should be correct set (131ms)
✓ Should fail if caller not owner (242ms)

removeSupportedToken
- ✓ Should fail if caller not owner (263ms)
- ✓ Should fail if Token is not supported (356ms)
- ✓ Should correct delete and emit event (638ms)

addSupportedToken
- ✓ Should fail if caller not owner (244ms)
- ✓ Should fail if fee is more or equal then 100% (220ms)
- ✓ Should fail token type out of range (248ms)
- ✓ Should correct emit event on add and set coorect field (496ms)
- ✓ Should correct emit and change field if add before (443ms)

addLiquidity
- ✓ Should fail if expected two amount on input (263ms)
- ✓ Should fail if amount is not bigger than zero (221ms)
- ✓ Should fail if token is not supported (261ms)
- ✓ Should correct transfer amount if native (341ms)
- ✓ Should correct transfer amount if token (127ms)
- ✓ Should correct change liquidity field (493ms)
- ✓ Should correct emit event

withdrawLiquidity
- ✓ Should fail if amount is zero (262ms)
- ✓ Should fail if amount is more then liquidity amount (250ms)
- ✓ Should correct withdraw native and emit event (225ms)
- ✓ Should correct withdraw tokens and emit event (645ms)

deposite

Should revert if
- ✓ token is not supported (274ms)
- ✓ amount is zero when token is not native (163ms)
- ✓ two amount is set (129ms)
- ✓ amount is zero when token is native (411ms)
- ✓ native not enougth (167ms)

Should correct deposite
- ✓ native balance sc is empty
- ✓ erc20 sc is empty (302ms)

native
- ✓ Should correct emit event
- ✓ Should correct transfer native to sc
- ✓ Should correct transfer native from user

erc20 type transfer only

```
        ✓ Should correct emit event
        ✓ Should correct transfer erc20 to sc (342ms)
        ✓ Should correct transfer native from user
      erc20 type mint/burn v1
        ✓ Should correct emit event
        ✓ Should correct burn erc20 (304ms)
        ✓ Should correct transfer native from user
      erc20 type mint/burn v2
        ✓ Should correct emit event
        ✓ Should correct transfer erc20 to sc (322ms)
        ✓ Should correct transfer native from user
    Should correct deposite if kyc needed and have fee
      native
        ✓ invalid signer when kyc is nedeed (276ms)
        ✓ Should correct emit event
        ✓ Should correct transfer native to sc
        ✓ Should correct transfer native from user
      erc20 with kec and fee
        ✓ invalid signer when kyc is nedeed (268ms)
        ✓ Should correct emit event
        ✓ Should call fee distributor (137ms)
        ✓ Should correct transfer erc20 to sc (271ms)
  Withdraw
    Should correct withdraw
      native
        ✓ Should correct emit event
        ✓ Should correct transfer native from sc
        ✓ Should correct transfer native to user
      erc20 type withdraw transfer only
        ✓ Should correct emit event
        ✓ Should correct transfer erc20 to sc (361ms)
        ✓ Should correct transfer native from user (39ms)
      erc20 type mint/burn v1/v2
        ✓ Should correct emit event
        ✓ Should correct burn erc20 (294ms)
        ✓ Should correct transfer native from user
    Should revert if
      ✓ nonce used before (236ms)
```

```
      ✓ invalid signer (578ms)
      ✓ erc20 amount not enoufg if erc20 type (353ms)
      ✓ native not enougth (315ms)
      ✓ token is not supported (184ms)
      ✓ Arrays have different lengths (779ms)
    Withdraw with multi sign
      Should correct withdraw
        native
          ✓ Should correct emit event
          ✓ Should correct transfer native from sc
          ✓ Should correct transfer native to user
      Should revert if
        ✓ nonce used before (270ms)
        ✓ invalid signer (599ms)
        ✓ Arrays have different lengths (965ms)

Contract: VestingFactory
  Functions
    createVesting
      ✓ Should fail if user not owner (772ms)
      ✓ Should correct transfer ownership (695ms)
    list
      ✓ Should return empty (99ms)
    add
      ✓ Should fail if user not have permisisons (325ms)
      ✓ Should add if all ok (311ms)
      ✓ Should emit event (297ms)
      ✓ Should fail if exists for user (497ms)
    remove
      ✓ Should fail if user not owner (241ms)
      ✓ Should remove if all ok (463ms)

Contract: Vesting
  ✓ GetInfo (137ms)
  getAvailAmountToDeposit
    ✓ Should return 0, 0 if all allocation used (86ms)
    ✓ Should get correct if remaning is less the max allocaiton (590ms)
    ✓ Should success (514ms)
```

completeVesting
- ✓ Should revert if caller not owner (179ms)
- ✓ Should revert if vesting can't be started (224ms)
- ✓ Should revert if Withdraw funds was called before (582ms)
- ✓ No need to burn tokens when everything is sold (2524ms)
- ✓ Should correct transfer amount (1054ms)
- ✓ Should correct burn amount (1068ms)

addDepositeAmount
- ✓ Should revert if caller not owner (264ms)
- ✓ Should revert if arr length incorrect (194ms)
- ✓ Should revert if vesting is started (443ms)
- ✓ Should fail if not have enough allocation (379ms)
- ✓ Should correct set amount (797ms)
- ✓ Should correct change totalDeposited (587ms)

Deposite
- ✓ Should be fail if incorrect sign (296ms)
- ✓ Should be fail if nonce used before (260ms)
- ✓ Should be fail if vesting is started (660ms)
- ✓ Should be can't transfer if is fiat (480ms)
- ✓ Check correct transfer (249ms)
- ✓ Check correct deposited (93ms)
- ✓ Check correct totalDeposited (492ms)
- ✓ Check set nonce (86ms)

  Should be fail if incorrect amount
  - ✓ if more then max allocation (234ms)
  - ✓ if more then min allocation (379ms)
  - ✓ if more then remaining allocation (310ms)
  - ✓ if more then remaining allocatio (2053ms)

Deployed
- ✓ signer (111ms)
- ✓ Should revert if total supply incorrect (795ms)
- ✓ Should revert if rewardToken incorrect (316ms)
- ✓ Should revert if deposite token incorrect (413ms)
- ✓ Should revert if signer incorrect (366ms)
- ✓ Should revert if token price incorrect (684ms)
- ✓ Should revert if allocation incorrect (907ms)

Set TGE
- ✓ Should revert if TGE is zero (193ms)

```
    ✓ Should revert if TGE is set (506ms)
    ✓ Should correct set TGE (367ms)
    ✓ Should revert if call not owner (231ms)
  getBalanceInfo
    ✓ Should return zero if zero deposited (215ms)
    ✓ Should return all balance how locked, if vesting don't start (88ms)
    ✓ Should return all unlocked balance if vesting started (350ms)
    ✓ Should return correct after harvest and by month linear (1800ms)
  harvest
    ✓ Should revert if vesting can't be started (248ms)
    ✓ Should correct transfer amount (735ms)
  Test case
    ✓ When VESTING_TYPE is SWAP (3710ms)
    ✓ When VESTING_TYPE is INTERVAL (11878ms)
    ✓ When one user bouth all tokens (10137ms)
    ✓ More user with specific case (8773ms)

  248 passing (3m)
```

Tests are written by Zokyo Security team

As part of our work assisting Gamestation in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the Gamestation contract requirements for details about issuance amounts and how the system handles these.

Code Coverage

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
contracts\	100.00	100.00	100.00	100.00	
GameStationBridge.sol	100.00	100.00	100.00	100.00	
All files	**100.00**	**100.00**	**100.00**	**100.00**	

Test Results

```
Contract: GameStationBridge
  constructor
    ✓ cannot deploy with incorrect fee (1003ms)
    ✓ cannot deploy with incorrect liquidity (504ms)
    ✓ cannot revert if signers array is empty (363ms)
    ✓ should revert if arrays have different lengths (428ms)
  check functions:
    IS_TOKEN_SUPPORTED
    ✓ should check supported token correctly (272ms)
    getSupportedTokens
    ✓ should return supported tokens correctly (112ms)
    addSigners
    ✓ should return supported tokens correctly (477ms)
  removeSigners
```

```
    ✓ should remove signers correctly (493ms)
    ✓ should revert if signer not found (278ms)
    ✓ should revert if signer not found (267ms)
    ✓ should revert if signers array is empty (573ms)
setFeeRecipient
    ✓ should set feeRecipient correctly (432ms)
setFeeDistributor
    ✓ should set feeDistributor correctly (376ms)
addLiquidity
    ✓ should add liquidity correctly (984ms)
    ✓ should revert if input two amount (218ms)
    ✓ should revert if amount is zero (209ms)
withdrawLiquidity
    ✓ should withdraw correctly (983ms)
    ✓ should withdraw correctly (NATIVE) (879ms)
    ✓ should revert if amount is zero (227ms)
    ✓ should revert if withdraw amount is greater than available amount (305ms)
    ✓ should catch event (560ms)
addSupportedToken
    ✓ should add supported token correctly (439ms)
    ✓ should revert if fee more than 99 (281ms)
    ✓ should catch event
removeSupportedToken
    ✓ should remove supported token correctly (490ms)
    ✓ should revert if token is not supported (228ms)
deposite
    ✓ should revert if token is not supported (1233ms)
    ✓ should revert if token is not supported (238ms)
    ✓ should revert if input two amount (245ms)
    ✓ should revert if amount is zero (326ms)
    ✓ should revert if invalid signer (359ms)
    ✓ should deposite correctly with kyc (ERC20) (1758ms)
    ✓ should catch event
    ✓ should deposite correctly with kyc (ERC20_MINT_BURN_V2) (2082ms)
    ✓ should deposite correctly with kyc (NATIVE/ERC20) (555ms)
withdraw
    ✓ should withdraw correctly (NATIVE) (448ms)
    ✓ should catch event
```

```
✓ should withdraw correctly (ERC20) (599ms)
✓ should withdraw correctly (ERC20_MINT_BURN) (695ms)
✓ should withdraw correctly (ERC20_MINT_BURN_V2) (734ms)
✓ should revert if nonce used before (288ms)
✓ should revert if invalid signer (316ms)
✓ should revert if token is not supported (297ms)
✓ should revert if token is not supported (203ms)

44 passing (27s)
```

We are grateful to have been given the opportunity to work with the Gamestation team.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that the Gamestation team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.