24th MARCH 2021

SMART CONTRACT AUDIT REPORT

-

version v2.0

Smart Contract Security Audit and General Analysis

HAECHI AUDIT

Table of Contents

2 Issues (1 Critical, 0 Major, 1 Minor) Found

About HAECHI AUDIT

HAECHI AUDIT is a global leading smart contract security audit and development firm operated by HAECHI LABS. HAECHI AUDIT consists of professionals with years of experience in blockchain R&D and provides the most reliable smart contract security audit and development services.

So far, based on the HAECHI AUDIT's security audit report, our clients have been successfully listed on the global cryptocurrency exchanges such as Huobi, Upbit, OKEX, and others.

Our notable portfolios include SK Telecom, Ground X by Kakao, and Carry Protocol while HAECHI AUDIT has conducted security audits for the world's top projects and enterprises.

Trusted by the industry leaders, we have been incubated by Samsung Electronics and awarded the Ethereum Foundation Grants and Ethereum Community Fund.

Contact : audit@haechi.io
Website : audit.haechi.io

01. Introduction

This report was written to provide a security audit for the KlaySwap smart contract. HAECHI AUDIT conducted the audit focusing on whether KlaySwap smart contract is designed and implemented in accordance with publicly released information and whether it has any security vulnerabilities.

The issues found are classified as CRITICAL, MAJOR, MINOR or TIPS according to their severity.

CRITICAL	Critical issues are security vulnerabilities that MUST be addressed in order to prevent widespread and massive damage.
MAJOR	Major issues contain security vulnerabilities or have faulty implementation issues and need to be fixed.
MINOR	Minor issues are some potential risks that require some degree of modification.
TIPS	Tips could help improve the code's usability and efficiency

02. Summary

The code used for the audit can be found at GitHub

- https://github.com/KlaySwap/contract-v2)
  commit hash: 6e5c524cf6cb3348b7d45c31d38a15cfa044b90f

Update

- v 2.0 commit hash: 618ca1dd640e7a0d658f89f46361c9a61b696518

Issues

HAECHI AUDIT has 1 Critical Issues, 0 Major Issues, and 1 Minor Issue; also, we included 2 Tip category that would improve the usability and/or efficiency of the code.

Severity	Issue	Status
CRITICAL	In some cases, Kai token can be borrowed regardless of the owned collateral.	(Found - v.1.0) (Resolved - v2.0)
MINOR	In some cases, rewards for KSP with the same lockPeriod are calculated differently.	(Found - v.1.0) (Intended - v2.0)
TIPS	Functions that perform the same function are implemented in duplicate.	(Found - v.1.0) (Intended - v2.0)
TIPS	An integer overflow may occur.	(Found - v.1.0) (Resolved - v2.0)

Update

[v2.0] - Ozys Team confirmed that one issue and one Tip were intended, and one issue and one Tip have been resolved.

03. Overview

Contracts Subject to Audit

- kai
    - Admin.impl.sol
    - Admin.sol
    - Kai.sol
- supporter
    - Supporter.impl.sol
    - Supporter.sol
    - Wallet.impl.sol
    - Wallet.sol
- treasury
    - Distribution.impl.sol
    - Distribution.sol
    - Treasury.impl.sol
    - Treasury.sol
- vksp
    - Governance.impl.sol
    - Governance.sol
    - PoolVoting.impl.sol
    - PoolVoting.sol
    - Store.impl.sol
    - Store.sol

        - VotingKSP.impl.sol
        - VotingKSP.sol
    - KSStore.sol
    - KlaytnExchange.impl.sol
    - KlaytnExchange.sol
    - KlaytnFactory.impl.sol
    - KlaytnFactory.sol
    - KlaytnMiningView.sol

Roles

KlaySwap Smart contract includes the following authorities.

- **Owner**
- **Liquidator**
- **Supporter**
- **Treasury**
- **Operator**
- **Executor**
- **Wallet**
- **ImplAdmin**
- **Factory**

The functions accessible by each authority are as follows.

Role	Functions
Owner	*Admin.impl.sol#changeNextOwner()* *Admin.impl.sol#setLiquidator()* *Admin.impl.sol#setCollateral()* *Kai.sol#transferOwnership()* *Kai.sol#changeNextOwner()* *Kai.sol#mint()* *Kai.sol#burn()* *Supporter.impl.sol#_changeNextOwner()* *Supporter.impl.sol#_setApporve()* *Supporter.impl.sol#_setDelegationContract()* *Supporter.sol#_setImplementation()* *Supporter.sol#_setWalletImplementation()* *Treasury.impl.sol#changeNextOwner()* *Treasury.impl.sol#changeCreationFee()* *Treasury.impl.sol#setOperator()* *Governance.impl.sol#changeNextOwner()* *Governance.impl.sol#addTransaction()* *Governance.impl.sol#changeNextOwner()* *KlaytnFactory.impl.sol#changeNextOwner()* *KlaytnFactory.impl.sol#changeCreateFee()* *KlaytnFactory.impl.sol#changeTeamWallet()* *KlaytnFactory.impl.sol#changePoolFee()* *KlaytnFactory.impl.sol#changeMiningRate()* *KlaytnFactory.impl.sol#sendReward()*
Liquidator	*Admin.impl.sol#liquidationCall()*
Supporter	*Wallet.impl.sol#grabLP()* *Wallet.impl.sol#claimKSP()* *Wallet.impl.sol#claimToken()*
Treasury	*Distribution.impl.sol#init()* *Distribution.impl.sol#depositKlay()* *Distribution.impl.sol#depositToken()* *Distribution.impl.sol#refixBlockAmount()* *Distribution.impl.sol#refixDistributionRate()* *Distribution.impl.sol#removeDistribution()* *Distribution.impl.sol#distribute()*
Operator	*Treasury.impl.sol#createKlayDistribution()* *Treasury.impl.sol#depositKlay()* *Treasury.impl.sol#depositToken()* *Treasury.impl.sol#refixBlockAmount()*

	Treasury.impl.sol#refixDistributionRate()
Executor	*Governance.impl.sol#setExecutor()* *Governance.impl.sol#init()* *Governance.impl.sol#executeTransaction()*
Wallet	*Governance.impl.sol#setKaiAdmin()* *Governance.impl.sol#setFeeShareRate()* *Governance.impl.sol#setVotingKSPMiningRate()* *Governance.impl.sol#setMaxMiningPoolCount()* *Governance.impl.sol#setStoreContract()* *Governance.impl.sol#setMaxMiningPoolCount()*
ImplAdmin	*Governance.impl.sol#setImplAdmin()* *Governance.sol#_setImplementation()* *Governance.sol#_setFactoryImplementation()* *Governance.sol#_setExchangeImplementation()* *Governance.sol#_setVotingKSPImplementation()* *Governance.sol#_setPoolVotingImplementation()* *Governance.sol#_setStoreImplementation()*
Factory	*KSStore.sol#setEntered()* *KSStore.sol#setNotEntered()* *KlaytnExchange.sol#grabKlayFromFactory()* *KlaytnExchange.sol#initPool()* *KlaytnExchange.sol#changeMiningRate()* *KlaytnExchange.sol#changeFee()* *KlaytnExchange.sol#exchangePos()* *KlaytnExchange.sol#exchangeNeg()*

04. Issues Found

Critical : In some cases, Kai token can be borrowed regardless of the owned collateral (Found - v.1.0) (Resolved - v.2.0)

CRITICAL

```
254.    function borrow(uint amount) public nonReentrant {
255.        require(amount != 0);
256.
257.        uint collateral = getCollateral(msg.sender);
258.        uint loan = getLoan(msg.sender);
259.
260.        require(loan <= collateral);
261.
262.        borrowBalance[msg.sender] = safeAdd(borrowBalance[msg.sender], amount);
263.        require(IKIP7(kai).mint(msg.sender, amount));
264.
265.        emitUserStat(msg.sender);
266.        emit Borrow(msg.sender, amount);
267.    }
268.
```

Problem Statement

AdminImpl#borrow() function lends Kai token to msg.sender. At this time, the contract must confirm whether the msg.sender is currently able to borrow Kai token.

However, according to the code implemented in the contract, only the past borrowings of msg.sender are compared in *AdminImpl#borrow()* by using *AdminImpl#getCollateral()* function and *AdminImpl#getLoan()* function. No syntax limits the amount of Kai token to be newly borrowed.

Thus, users who have not borrowed Kai in the past or a very small amount of it can borrow a larger amount of Kai token than the collateral held by using *AdminImpl#borrow()* function.

Example

0 . Suppose a user without borrowed Kai token uses the borrow function to borrow Kai token.

1. Because borrowBalance[user] = 0, the loan value of the user becomes 0.

2. Thus, regardless of the user's collateral value, it always passes the require syntax in line 260.

3. As such, the user can borrow a large amount of Kai token regardless of collateral possession.

Recommendation

We recommend limiting the act of borrowing a larger amount of Kai token than the collateral owned by adding a syntax that compares the loan after changes in borrowBalance and collateral of msg.sender.

Update

[v2.0] - Ozys Team modified the code to confirm the comparison between loan and collateral after an increase in borrowBalance and resolved the above issue.

Minor : In some cases, rewards for KSP with the same lockPeriod are calculated differently (Found - v.1.0) (Intended - v.2.0)

MINOR

```
150.    function lockKSP(uint amount, uint lockPeriodRequested) public nonReentrant {
151.        require(amount != 0);
152.        amount = safeMul(amount, 10 ** 18);
153.
154.        require(lockPeriodRequested == 120 days || lockPeriodRequested == 240 days ||
        lockPeriodRequested == 360 days);
155.        giveReward(msg.sender);
156.
157.        uint mintAmount = 0;
158.        if(lockPeriodRequested == 120 days){
159.            mintAmount = amount;
160.        }
161.        if(lockPeriodRequested == 240 days){
162.            mintAmount = safeMul(amount, 2);
163.        }
164.        if(lockPeriodRequested == 360 days){
165.            mintAmount = safeMul(amount, 4);
166.        }
167.        require(mintAmount != 0);
168.        lockedKSP[msg.sender] = safeAdd(lockedKSP[msg.sender], amount);
169.        balanceOf[msg.sender] = safeAdd(balanceOf[msg.sender], mintAmount);
170.        totalSupply = safeAdd(totalSupply, mintAmount);
171.        emit Transfer(address(0), msg.sender, mintAmount);
172.
173.        if(safeAdd(now, lockPeriodRequested) > unlockTime[msg.sender]){
174.
175.            unlockTime[msg.sender]= safeAdd(now, lockPeriodRequested);
176.        }
177.
178.        if(lockPeriod[msg.sender] < lockPeriodRequested){
179.
180.            lockPeriod[msg.sender] = lockPeriodRequested;
181.        }
182.
183.        addSnapShot(msg.sender);
184.
185.        emit LockKSP(msg.sender, lockPeriodRequested, amount);
186.    }
187.
```

Problem Statement

VotingKSPImpl#lockKSP() function creates rewards when users lock their KSP token to the contract. At this time, the locked KSP is stored in lockedKSP[msg.sender] and the resulting reward in balanceOf[msg.sender], which are later distributed to users through *VotingKSPImpl#giveReward()* function.

However, when a user's asset is already locked and the user additionally locks assets, the mintAmount of the asset with the same lockPeriod is set differently, causing the user's rewards to be distributed differently afterward.

Example

0 . Suppose that 100 days have elapsed after users A and B locked 100 KSP as lockPeriodRequested : 360days.

1. User A additionally locks 100 KSP as lockPeriodRequested : 120days. At this time, a value of 100 is added to balanceOf[A], making the lockPeriod of 200 KSP in total as 360 days.

2. User B additionally locks 100 KSP as lockPeriodRequested : 240days. At this time, a value of 200 is added to balanceOf[B], making the lockPeriod of 200 KSP in total as 360 days.

3. Both users initially locked 100 KSP and an additional 100 KSP after 100 days. After 360 days since the initial locking, the amounts of rewards for 200 KSP that users A and B would be distributed upon unlocking will differ.

Recommendation

The above situation does not have a possibility that damage may be caused by a malicious user. However, if this implementation is unintended, it is advised to use logic that pays the same amount of rewards for the KSPs with the same lockPeriod.

Update

[v2.0] - Ozys Team implemented the code so that users can freely select lockPeriod for additional locking and confirmed that the above issue is an intended implementation.

TIPS : Functions that perform the same function are implemented in duplicate (Found - v.1.0) (Intended - v.2.0)

TIPS

```
188.  function transferOwnership(address newOwner) public {
189.      require(msg.sender == _owner);
190.      require(newOwner != address(0));
191.
192.      _owner = newOwner;
193.      _nextOwner = address(0);
194.  }
195.
196.  function changeNextOwner(address nextOwner) public {
197.      require(msg.sender == _owner);
198.      require(nextOwner != address(0));
199.
200.      _nextOwner = nextOwner;
201.  }
202.
203.  function changeOwner() public {
204.      require(msg.sender == _nextOwner);
205.
206.      _owner = _nextOwner;
207.      _nextOwner = address(0);
208.  }
209.
```

Both of *KIP7#transferOwnership()* function and *KIP7#changeOwner(), KIP7()#changeNextOwner()* functions change the owner of the contract. At this time, there is a feature that newOwner's approval is not required when *KIP7#transferOwnership()* function is used to change the owner.

Thus, unless there is a case of changing the owner without the approval of newOwner, we recommend deleting the above function.

Update

[v2.0] - Ozys Team implemented both methods of changing the owner for convenience and confirmed the above issue is an intended implementation.

TIPS : An integer overflow may occur (Found - v.1.0) (Resolved - v.2.0)

TIPS

```
288.  function setMiningRate() public {
289.      require(vKSPMiningRate != 0);
290.
291.      PoolVotingLike pv = PoolVotingLike(poolVoting);
292.
293.      uint poolCount = pv.poolCount();
294.      require(poolCount > 0);
295.
296.      uint len;
297.      if(poolCount > MAX_MINING_POOL_COUNT){
298.          len = MAX_MINING_POOL_COUNT + 1;
299.      }else{
300.          len = poolCount + 1;
301.      }
302.
303.      uint i;
304.      uint sum = 0;
305.      for(i=1; i<len; i++){
306.          sum = sum + pv.poolAmount( pv.poolRanking(i-1) );
307.      }
```

```
229.  function mined() public view returns (uint) {
230.      uint nowBlock = block.number;
231.
232.      if (nowBlock < minableBlock) return 0;
233.      uint level = (nowBlock + 1 - minableBlock) / halfLife;
234.      uint elapsed = (nowBlock + 1 - minableBlock) % halfLife;
235.
236.      uint num = 0;
237.      uint den = 1;
238.
239.      if (level == 0) {
240.          num = (100 - teamRatio) * elapsed;
241.          den = 200 * halfLife;
242.      }
243.      else if (level == 1) {
244.          num = (100 - teamRatio) * halfLife + 50 * elapsed;
245.          den = 200 * halfLife;
246.      }
247.      else {
248.          num = (150 - teamRatio) * halfLife;
249.          den = 200 * halfLife;
250.
251.          for (uint l = 3; l <= level; l++) {
252.              num = safeAdd(safeMul(num, 2), 50 * halfLife);
253.              den = safeMul(den, 2);
254.          }
255.
256.          num = safeAdd(num, 25 * elapsed);
257.      }
258.
259.      return miningAmount * num / den;
260.  }
261.
```

In the syntaxes in line 306 of the above *GovernanceImpl#setMiningRate()* and in line 259 of *FactoryImpl#mined()*, integer overflow may occur.

When conducting calculations on uint, it is advised to use safeMath library to prevent integer overflow.

Update

[v2.0] - Ozys Team modified the code to calculate uint by using safeMath in the above *GovernanceImpl#setMiningRate()* , *FactoryImpl#mined()* and solved the issue.

05. Disclaimer

This report is not an advice on investment, nor does it guarantee adequacy of a business model and/or a bug-free code. This report should be used only to discuss known technical problems. The code may include problems on Ethereum that are not included in this report. It will be necessary to resolve addressed issues and conduct thorough tests to ensure the safety of the smart contract.

Appendix A. Test Results

The results below show the unit test results that cover the main logic of the smart contract subject to the security audit. The parts marked in red are test cases that failed to pass the test due to issues.

```
AdminImpl
  version()
    ✓ should return version
  changeNextOwner()
    ✓ should be reverted when msg.sender is not owner
    ✓ should change nextOwner
  changeOwner()
    ✓ should be reverted when msg.sender is not nextOwner
    ✓ should change owner
  setLiquidator()
    ✓ should be reverted when msg.sender is not nextOwner
    ✓ should be reverted when liquidator is zeroAddr
    ✓ should set liquidator
  setCollateral()
    ✓ should be reverted when borrowFactor <= 10000
    ✓ should be reverted when liquidationFactor <= 10000
   valid case
     ✓ should update borrowFactors
     ✓ should update liquidationFactors
  setOperatorApproval()
    ✓ should store appropriate operatorApprovals
    ✓ should emit UserStat when user borrowed kai (174ms)
  borrow()
    ✓ should be reverted when borrow amount is zero
    ✓ should be reverted when liquidationFactor <= 10000
   1) should fail when transfer collateral after borrow
   2) should fail when borrow over collateral
   valid case
     ✓ should update borrowBalance (90ms)
     ✓ should mint kai token (88ms)
     ✓ should emit Borrow event (81ms)
  repay()
    ✓ should be reverted when repay amount is zero
    ✓ should be reverted when now borrowed before
   valid case
```

```
    ✓ should update borrowBalance (57ms)
    ✓ should burn kai token (56ms)
    ✓ should update operatorApprovals (62ms)
    ✓ should emit Repay event (62ms)
  liquidationCall()
    ✓ should fail when msg.sender is not liquidator
    ✓ should be reverted when user is address zero
    ✓ should be reverted when now borrowed before
    valid case
      ✓ should transfer token to liquidator (93ms)
      ✓ should update liquidationBalance (122ms)
      ✓ should update borrowBalance (101ms)
      ✓ should update operatorApprovals (96ms)
      ✓ should emit LiquidationCall event (94ms)
  liquidate()
    ✓ should be reverted when amount is zero
    ✓ should be reverted when no liquidate before
    ✓ should be reverted when amount is larger than liquidationBalance (258ms)
    valid case
      ✓ should burn token
      ✓ should update liquidationBalance
      ✓ should emit Liquidate event
  canRemoveApprovals()
    ✓ should return true when initialized
    CASE : not initialized
      ✓ should return appropriate result
  canTransferable()
    ✓ should return true when initialized
    CASE : not initialized
      ✓ should return appropriate result (39ms)

KaiStablecoin
  constructor()
    ✓ should return appropriate name
    ✓ should return appropriate symbol
    ✓ should return appropriate decimals
  KIP7Spec()
    #approve()
      ✓ should fail if spender is AddressZero
      valid case
        ✓ allowance should set appropriately
        ✓ should emit Approval event
```

```
  #transfer()
    ✓ should fail if recipient is AddressZero
    ✓ should fail if sender's amount is more than balance
    valid case
      ✓ sender's balance should decrease
      ✓ receipient's balance should increase
      ✓ should emit Transfer event
    valid case : safeTransfer
      ✓ sender's balance should decrease
      ✓ receipient's balance should increase
      ✓ should emit Transfer event
  #transferFrom()
    ✓ should fail if sender is AddressZero
    ✓ should fail if recipient is AddressZero
    ✓ should fail if sender's amount is more than balance
    ✓ should fail if sender's amount is more than allowance (47ms)
    ✓ should fail if try to transfer sender's token without approve process
    valid case
      ✓ sender's balance should decrease
      ✓ receipient's balance should increase
      ✓ allowance should decrease
      ✓ should emit Transfer event
      ✓ should emit Approval event
    valid case : safeTransferFrom
      ✓ sender's balance should decrease
      ✓ receipient's balance should increase
      ✓ allowance should decrease
      ✓ should emit Transfer event
      ✓ should emit Approval event
  #burn()
    ✓ should fail if try to burn more than burner's balance
    valid case
      ✓ account's balance should decrease
      ✓ totalSupply should decrease
      ✓ should emit Transfer event
      3) should emit Burn event
Ownable Spec()
  transferOwnership()
    ✓ should be reverted when msg.sender is not owner
    ✓ should be reverted when newOwner is zeroAddr
    ✓ should change owner
  changeNextOwner()
```

```
    ✓ should be reverted when msg.sender is not owner
    ✓ should be reverted when newOwner is zeroAddr
    ✓ should change nextOwner
  changeOwner()
    ✓ should be reverted when msg.sender is not nextOwner
    ✓ should change owner

ExchangeImpl
  version()
    ✓ should return version
  #claimReward()
    ✓ should claim treasury (40ms)
    ✓ should update userLastIndex
    ✓ should send reward
    ✓ should update userRewardSum (38ms)
    ✓ should emit GiveReward event
  #decreaseBalance()
    ✓ should emit GiveReward event (44ms)
    ✓ should decrease balance (43ms)
    ✓ should set operator approval (45ms)
  #increaseBalance()
    ✓ should emit GiveReward event
    ✓ should increase balance (45ms)
    ✓ should emit user stat (44ms)
  #transfer()
    ✓ should increase recipient balance (84ms)
    ✓ should decrease user balance (71ms)
    ✓ should emit Transfer event (77ms)
  #transferFrom()
    ✓ should increase recipient balance (76ms)
    ✓ should decrease user balance (98ms)
    ✓ should decrease allowance (91ms)
    ✓ should emit Transfer event (94ms)
  #approve()
    ✓ should fail when spender is address(0)
    ✓ should increase allowance
    ✓ should emit Approval event
  #setOperatorApproval()
    ✓ should fail when kaiAdmin is address(0)
    ✓ should set operator approval
  getMiningIndex()
    ✓ should return appropriate value
```

```
updateMiningIndex()
  ✓ should update lastMined
  ✓ should update miningIndex
  ✓ should emit UpdateMiningIndex event
increaseTotalSupply()
  ✓ should update lastMined
  ✓ should update miningIndex
  ✓ should increase totalSupply
decreaseTotalSupply()
  ✓ should update lastMined
  ✓ should update miningIndex
  ✓ should decrease totalSupply
changeMiningRate()
  ✓ should fail when mining > 100
  ✓ should update lastMined
  ✓ should update miningIndex
  ✓ should update mining
  ✓ should emit ChangeMiningRate event
changeFee()
  ✓ should fail when fee > 100
  ✓ should fail when msg.sender is not factory
  ✓ should update fee
  ✓ should emit ChangeFee event
getCurrentPool()
  ✓ should return contract's klaytn balance when tokenA = 0
  ✓ should return contract's tokenA balance when tokenA != 0
estimatePos()
  ✓ should fail when token is not tokenA or B
  ✓ should return appropriate value when token = tokenA
  ✓ should return appropriate value when token = tokenB
estimateNeg()
  ✓ should fail when token is not tokenA or B
  ✓ should return appropriate value when token = tokenA
  ✓ should return appropriate value when token = tokenB
grabToken()
  ✓ should fail when token is address(0)
  ✓ should increase contract token balance (50ms)
  ✓ should decrease user token balance (45ms)
sendToken()
  CASE : token is address(0)
    ✓ should increase user klaytn balance
    ✓ should decrease contract klaytn balance
```

```
  CASE : token is not address(0)
    ✓ should increase user token balance
    ✓ should decrease contract token balance
sendTokenToPoolVoting()
  CASE : token is address(0)
    ✓ should grabKlayFromExchange
  CASE : token is not address(0)
    ✓ should increase pvoting token balance
    ✓ should decrease contract token balance
exchangePos()
  ✓ should fail when token is not tokenA or tokenB
  ✓ should fail when amount is 0
  CASE : token is tokenA
    ✓ should send token to exchange (84ms)
    ✓ should send token (93ms)
    ✓ should emit ExchangePos (89ms)
    ✓ should send token to poolVoting (92ms)
    ✓ should update market (90ms)
  CASE : token is tokenB
    ✓ should send token to exchange (86ms)
    ✓ should send token (91ms)
    ✓ should emit ExchangePos (92ms)
    ✓ should send token to poolVoting (92ms)
    ✓ should update market (82ms)
exchangeNeg()
  ✓ should fail when token is not tokenA or tokenB
  ✓ should fail when amount is 0
  CASE : token is tokenA
    ✓ should send token to exchange (88ms)
    ✓ should send token (98ms)
    ✓ should emit ExchangeNeg (82ms)
    ✓ should send token to poolVoting (96ms)
    ✓ should update market (84ms)
  CASE : token is tokenB
    ✓ should send token to exchange (90ms)
    ✓ should send token (97ms)
    ✓ should emit ExchangeNeg (78ms)
    ✓ should send token to poolVoting (110ms)
    ✓ should update market (89ms)
addLiquidity()
  CASE : tokenA is not address(0)
    ✓ should fail when amount is zero
```

```
    case : first LP
      ✓ should grab tokenA (108ms)
      ✓ should grab tokenB (105ms)
      ✓ should increase total supply (100ms)
      ✓ should increase balance of msg.sender (93ms)
      ✓ should emit AddLiquidity event (105ms)
    case : not first LP
      case : A < B
        ✓ should grab tokenA (144ms)
        ✓ should grab tokenB (125ms)
        ✓ should increase total supply (110ms)
        ✓ should increase balance of msg.sender (108ms)
        ✓ should emit AddLiquidity event (116ms)
      case : A > B
        ✓ should grab tokenA (120ms)
        ✓ should grab tokenB (157ms)
        ✓ should increase total supply (101ms)
        ✓ should increase balance of msg.sender (92ms)
        ✓ should emit AddLiquidity event (104ms)
  CASE : tokenA is address(0)
    case : first LP
      ✓ should grab tokenB (97ms)
      ✓ should increase total supply (96ms)
      ✓ should increase balance of msg.sender (80ms)
      ✓ should emit AddLiquidity event (103ms)
    case : not first LP
      case : A < B
        ✓ should grab tokenB (109ms)
        ✓ should increase total supply (82ms)
        ✓ should increase balance of msg.sender (89ms)
        ✓ should emit AddLiquidity event (87ms)
      case : A > B
        ✓ should grab tokenB (103ms)
        ✓ should increase total supply (78ms)
        ✓ should increase balance of msg.sender (82ms)
        ✓ should transfer left Klaytn to msg.sender (82ms)
        ✓ should emit AddLiquidity event (77ms)
removeLiquidity()
  ✓ should fail when amount is 0
  ✓ should decrease total supply (117ms)
  ✓ should decrease balance of msg.sender (96ms)
  ✓ should send tokenA to msg.sender (102ms)
```

```
    ✓ should send tokenB to msg.sender (101ms)
    ✓ should emit RemoveLiquidity event (91ms)
  addKlayLiquidityWithLimit()
    ✓ should fail when tokenA is not address(0)
    ✓ should fail when realA < minAmountA (188ms)
    ✓ should fail when realB < minAmountB (98ms)
  addKctLiquidityWithLimit()
    ✓ should fail when tokenA is address(0) (38ms)
    ✓ should fail when realA < minAmountA (371ms)
    ✓ should fail when realB < minAmountB (109ms)
  removeLiquidityWithLimit()
    ✓ should fail when amount is 0
    ✓ should fail when amountA < minAmountA (41ms)
    ✓ should fail when amountB < minAmountB
    ✓ should decrease total supply (83ms)
    ✓ should decrease balance of msg.sender (100ms)
    ✓ should send tokenA to msg.sender (100ms)
    ✓ should send tokenB to msg.sender (103ms)
    ✓ should emit RemoveLiquidity event (102ms)
  needAllowance()
    ✓ should return true when kaiAdmin is address(0)
  getTreasury()
    ✓ should return treasury address
  getTokenSymbol()
    ✓ should return KLAY when token is address(0)
    ✓ should return toke symbol
  initPool()
    ✓ should increase total supply (59ms)
    ✓ should increase balance of msg.sender (65ms)
    ✓ should update decimals (59ms)
  calcPos()
    ✓ should return 0 when totalSupply is 0
  calcNeg()
    ✓ should return maxUint when output > poolOut

FactoryImpl
  version()
    ✓ should return version
  KIP7Spec()
   #approve()
     ✓ should fail if spender is AddressZero
     valid case
```

```
    ✓ allowance should set appropriately
    ✓ should emit Approval event
  #transfer()
   4) should fail if recipient is AddressZero
   ✓ should fail if sender's amount is more than balance
   valid case
    ✓ sender's balance should decrease
    ✓ receipient's balance should increase
    ✓ should emit Transfer event
  #transferFrom()
   ✓ should fail if sender is AddressZero
   5) should fail if recipient is AddressZero
   ✓ should fail if sender's amount is more than balance
   ✓ should fail if sender's amount is more than allowance
   ✓ should fail if try to transfer sender's token without approve process
   valid case
    ✓ sender's balance should decrease
    ✓ receipient's balance should increase
    ✓ allowance should decrease
    ✓ should emit Transfer event
    6) should emit Approval event
changeNextOwner()
  ✓ should be reverted when msg.sender is not owner
  ✓ should change nextOwner
changeOwner()
  ✓ should be reverted when msg.sender is not nextOwner
  ✓ should change owner
changeCreateFee()
  ✓ should be reverted when msg.sender is not owner
  ✓ should change nextOwner
changeTeamWallet()
  ✓ should be reverted when msg.sender is not owner
  ✓ should change nextOwner
changePoolFee()
  ✓ should be reverted when msg.sender is not owner
  ✓ should be reverted when fee is over 100
  ✓ should be reverted when tokenPool is address(0)
  ✓ should change poolFee
changeMiningRate()
  ✓ should be reverted when msg.sender is not owner
  ✓ should be reverted when token length is 1
  ✓ should be reverted when array length is different
```

```
  ✓ should fail when rate is 0
  ✓ should fail when excs is address(0)
  ✓ should fail when excs is same (41ms)
  ✓ should fail when rate[0] is not zero
  ✓ should fail when rateSum is not 100
 valid case
   ✓ should change mining rate (54ms)
   ✓ should set mining rate 0 when already has rate (53ms)
   ✓ should set mining rate 0 when already has rate (77ms)
mined()
  ✓ should return 0 when minableBlock < nowBlock
stepPos
  ✓ should fail when result = outAmount
stepNeg
  ✓ should fail when result = outAmount
estimatePos
  ✓ should fail when exc = address(0)
  ✓ should fail when outAmount is zero
  ✓ should return appropriate value
estimateNeg
  ✓ should fail when exc = address(0)
  ✓ should fail when inAmount is maxUint
  ✓ should return appropriate value
grabKlayFromExchange()
  ✓ should fail when pool not exist
grabToken()
  ✓ should decrease user balance
  ✓ should increase contract balance
sendKct()
  ✓ should increase user balance
  ✓ should decrease contract balance
sendToken()
  ✓ should increase user balance (39ms)
  ✓ should decrease contract balance
sendTokenToExchange()
  ✓ should fail when pool not exist
  ✓ should increase user balance
  ✓ should decrease contract balance
sendReward()
  ✓ should fail when msg.sender is not owner
  ✓ should decrease contract's balance
  ✓ should increase user's balance
```

```
  getPoolCount()
    ✓ should return pool length
  getPoolAddress()
    ✓ should return pool length
    ✓ should fail when invalid index
  exchangePos()
    ✓ should fail when amount is zero
    ✓ should fail when amountA < amountB
    ✓ should grab tokenA (91ms)
    ✓ should sned tokenB to user (124ms)
    ✓ should sned tokenB to user (81ms)
  exchangeNeg()
    ✓ should fail when amount is zero
    ✓ should fail when amountA < amountB (45ms)
    ✓ should grab tokenA (112ms)
    ✓ should sned tokenB to user (107ms)
    ✓ should sned tokenB to user (86ms)
  createPool()
    ✓ should fail when amountA and B is zero
    ✓ should fail when tokenToPool is not address(0)
    ✓ should decrease msg.sender's balance (61ms)
    ✓ should decrease totalSupply (63ms)
    ✓ should transfer tokenA user to exc (72ms)
    ✓ should transfer tokenB user to exc (62ms)
    ✓ should init Pool (69ms)
    ✓ should add pool (63ms)
    ✓ should update poolExist (55ms)
    ✓ should grabKlayFromFactory user to contract (52ms)

KSStore
  constructor()
    ✓ should store factory
    ✓ should store _entered
  setEntered()
    ✓ should fail when msg.sender is not factory
    ✓ should store _entered ture
  setNotEntered()
    ✓ should fail when msg.sender is not factory
    ✓ should store _entered ture

SafeMath
  add
```

```
    ✓ adds correctly
    ✓ reverts on addition overflow
  sub
    ✓ subtracts correctly
    ✓ reverts if subtraction result would be negative
  mul
    ✓ multiplies correctly
    ✓ multiplies by zero correctly
    ✓ reverts on multiplication overflow
  div
    ✓ divides correctly
    ✓ divides zero correctly
    ✓ returns complete number result on non-even division
    ✓ reverts on division by zero
  mod
    ✓ reverts with a 0 divisor
   modulos correctly
      ✓ when the dividend is smaller than the divisor
      ✓ when the dividend is equal to the divisor
      ✓ when the dividend is larger than the divisor
      ✓ when the dividend is a multiple of the divisor

SupporterImpl
  _changeNextOwner()
    ✓ should be reverted when msg.sender is not owner
    ✓ should change nextOwner
  _changeOwner()
    ✓ should be reverted when msg.sender is not nextOwner
    ✓ should change owner (38ms)
  _setApporve()
    ✓ allowance should set appropriately (123ms)
  _setDelegationContract()
    ✓ allowance should set appropriately
  _estimateKlayFromDelegation()
    ✓ should return appropriate value
  _estimateSKlayFromDelegation()
    ✓ should return appropriate value
  _estimateKlayFromSwap()
    ✓ should return appropriate value
  _estimateSKlayFromSwap()
    ✓ should return appropriate value
  _estimateKlayFromLiquidity()
```

```
  ✓ should return appropriate value
_estimateSKlayFromLiquidity()
  ✓ should return appropriate value
_estimateAddLiquidity()
 CASE : withKlay < withSKlay
   ✓ should return appropriate LP
   ✓ should return appropriate Klay
   ✓ should return appropriate SKlay
 CASE : withKlay >= withSKlay
   ✓ should return appropriate LP
   ✓ should return appropriate Klay
   ✓ should return appropriate SKlay
_estimateLPFromKlay()
  ✓ should return appropriate value
_estimateKlayFromLP()
  ✓ should return appropriate value
_estimateKlayByQuickWithdraw()
  ✓ should return appropriate klay
  ✓ should return appropriate sklay (40ms)
  ✓ should return appropriate klayFromSwap (42ms)
  ✓ should return appropriate totalReturn (38ms)
_calc()
  ✓ should return appropriate liquidityKlay (195ms)
  ✓ should return appropriate delegationKlay (192ms)
  ✓ should return appropriate swapKlay (182ms)
  ✓ should return appropriate totalSKlay (214ms)
_calcKlayForDelegation()
  ✓ should return appropriate value
_getPoolRate()
  ✓ should return appropriate delegationPoolRate
  ✓ should return appropriate swapPoolRate
_comparePoolRate()
  ✓ should return appropriate value
addLiquidity()
 valid case : 1
   ✓ should transfer token to wallet[msg.sender] (269ms)
   ✓ should emit AddLiquidity Event (240ms)
 valid case : 2
   ✓ should transfer token to wallet[msg.sender] (113ms)
   ✓ should emit AddLiquidity Event (91ms)
removeLiquidity()
  ✓ should fail when amount is zero
```

```
  CASE : amount is MAX256
    ✓ should emit RemoveLiquidity event (44ms)
    ✓ should deposit sklay (82ms)
    ✓ should unstakeKlayWithSklay (76ms)
 claimKlay()
  ✓ should fail when claimCounter > historyCounter
  ✓ should fail when claimCounter + 1 != hid
  ✓ should fail when already claimed (59ms)
  ✓ should fail when user is not equal h.user (55ms)
  ✓ should fail when not completed yet (52ms)
  CASE : beforeState is 0
    ✓ should claim unstaking klay
    ✓ should emit ClaimKlay Event (45ms)
  CASE : beforeState is not 0
    ✓ should emit ClaimCanceledKlay Event (63ms)
 version()
  ✓ should return version
 _getUserStat()
  ✓ should return wallet (42ms)
  ✓ should return lp
  ✓ should return klay
  ✓ should return ksp
  ✓ should return historyIndex
  ✓ should return rewardKSPSum
  ✓ should return lastKSPIndex
 _checkPoolRate()
  ✓ should fail when diff is larger than swapPoolRate/100
 claimKSP1()
  ✓ should fail when not unfreezed yet
  ✓ should claim KSP (41ms)
 claimToken()
  ✓ should fail when token is pool Contract
  ✓ should fail when token is factory Contract
  ✓ should claim KSP

WalletImpl
 version()
  ✓ should return version
 claimKSP()
  ✓ should fail when msg.sender is not supporter
  valid case
    ✓ should claim reward (219ms)
```

```
    ✓ should transfer KSP token to user (41ms)
  claimToken()
    ✓ should fail when msg.sender is not supporter
    ✓ should fail when token is KSPToken
    ✓ should fail when token is LPToken
    case : token is not zero_address
      ✓ should fail when contract has no token
      ✓ should transfer token (52ms)
    case : token is zero_address
      ✓ should transfer Klaytn
  grabLP()
    ✓ should fail when msg.sender is not supporter
    ✓ should fail when contract balance < amount
    valid case
      ✓ should transfer LPToken to supporter (55ms)

DistributionImpl
  version()
    ✓ should return version
  init()
    ✓ should fail when blockAmount is zero
    ✓ should fail when blockNumber is smaller than block.number
    ✓ should fail when target.length != rate.length
    ✓ should fail when rateSum is not 100
    ✓ should fail when already initialized (55ms)
    vallid case
      ✓ should emit Initialized event (38ms)
      ✓ should store token (47ms)
  depositKlay()
    ✓ should fail when msg.sender is not treasury
    ✓ should fail when token is not zeroAddress (60ms)
    ✓ should fail when msg.value is zero (64ms)
    vallid case
      ✓ should increase totalAmount
      ✓ should store distributedAmount (41ms)
      ✓ should emit Initialized event
  depositToken()
    ✓ should fail when msg.sender is not treasury
    ✓ should fail when token is zeroAddress (57ms)
    ✓ should fail when deposit amounut is zero (100ms)
    vallid case
      ✓ should increase totalAmount
```

```
    ✓ should store distributedAmount (64ms)
    ✓ should emit Initialized event
  refixDistributionRate()
    ✓ should fail when msg.sender is not treasury
    ✓ should fail when totalAmount is smaller than distribution
    vallid case
    ✓ should update distributedRate
    ✓ should update targetEntries
  refixBlockAmount()
    ✓ should fail when blockAmount is zero
    valid case
    ✓ should update distributionIndex[token] (56ms)
  distribute()
    CASE : token is not address(0)
    ✓ should transfer token to user
    ✓ should update userLastIndex
    ✓ should update userRewardSum
    CASE : token is address(0)
    ✓ should transfer klaytn (59ms)
  estimateEndBlock()
    ✓ should return appropriate value
  removeDistribution()
    ✓ should fail when not sufficient time passed
    CASE : token is address(0)
    ✓ should transfer klaytn
    CASE : token is not address(0)
    ✓ should transfer token (39ms)
  distribution()
    ✓ should return appropriate value
  getDistributionIndex()
    ✓ should return appropriate value (84ms)

TreasuryImpl
  version()
    ✓ should return version
  changeNextOwner()
    ✓ should be reverted when msg.sender is not owner
    ✓ should change nextOwner
  changeOwner()
    ✓ should be reverted when msg.sender is not nextOwner
    ✓ should change owner
  changeCreationFee()
```

```
  ✓ should be reverted when msg.sender is not owner
  ✓ should change Creation Fee
setOperator()
  ✓ should be reverted when msg.sender is not owner
  7) should fail when already valid operator
  ✓ should validate operator
createKlayDistribution()
  ✓ should be reverted when msg.sender is not operator
  ✓ should fail when targetNum != rateNum
  ✓ should fail when blockNumber is past
  ✓ should fail when blockAmount is zero
  valid case
    ✓ should init distribution
    ✓ should store distributions info
    ✓ should store distributionOperator info
    ✓ should deposit klay
    ✓ should store distributionEntries info
    ✓ should store distributionCount info
createTokenDistribution()
  ✓ should be reverted when msg.sender is not operator
  ✓ should fail when token is address(0)
  ✓ should fail when not approved token yet
  valid case
    ✓ should init distribution
    ✓ should store distributions info
    ✓ should store distributionOperator info
    ✓ should deposit token
    ✓ should store distributionEntries info
    ✓ should store distributionCount info
depositKlay()
  ✓ should be reverted when msg.sender is not operator
  valid case
    ✓ should deposit klay
    ✓ should emit DEPOSIT event
depositToken()
  ✓ should be reverted when msg.sender is not operator
  ✓ should fail when token is address(0)
  ✓ should fail when not approved token yet (38ms)
  valid case
    ✓ should deposit token (51ms)
    ✓ should emit DEPOSIT event (39ms)
refixBlockAmount()
```

```
    ✓ should be reverted when msg.sender is not operator
    ✓ should fail when block amount is zero
   valid case
    ✓ should refix block amount (39ms)
    ✓ should emit DEPOSIT event
  refixDistributionRate()
    ✓ should be reverted when msg.sender is not operator
    ✓ should fail when target != rate
   valid case
    ✓ should refix block rate
    ✓ should store distributionEntries info
    ✓ should store distributionCount info
    ✓ should emit DEPOSIT event
  removeDistribution()
    ✓ should fail when already removed (38ms)
   valid case
    ✓ should remove Distribution
    ✓ should store distributionOperator info
    ✓ should store distributions info
    ✓ should emit DEPOSIT event
  claim()
   valid case
    ✓ should store distributionEntries info
    ✓ should store distributionCount info
    ✓ should distribute user and target
  claims()
    ✓ should fail when msg.sender is not target
   valid case
    ✓ should store distributionEntries info
    ✓ should store distributionCount info

GovernanceImpl
  version()
    ✓ should return version
  changeNextOwner()
    ✓ should be reverted when msg.sender is not owner
    ✓ should change nextOwner
  changeOwner()
    ✓ should be reverted when msg.sender is not nextOwner
    ✓ should change owner
  setImplAdmin()
    ✓ should be reverted when msg.sender is not owner
```

```
  ✓ should change ImplAdmin
setExecutor()
  ✓ should be reverted when msg.sender is not owner
  ✓ should change ImplAdmin
init()
  ✓ should be reverted when msg.sender is not executor
  ✓ should be reverted when nextOwner is not Gov (40ms)
  ✓ should be reverted when voting.governance is not Gov (67ms)
  ✓ should be reverted when poolvoting.governance is not Gov (78ms)
  ✓ should be reverted when feeShareRate > 100 (94ms)
  ✓ should be reverted when already initialized (96ms)
 valid case
   ✓ should make isInitialized true
   ✓ should initialize factory
   ✓ should initialize votingKSP
   ✓ should initialize poolVoting
   ✓ should initialize treasury
   ✓ should initialize feeShareRate
setKaiAdmin()
  ✓ should be reverted when msg.sender is not owner
  ✓ should be reverted when kaiAdmin is address(0)
  ✓ should change kaiAdmin
setFeeShareRate()
  ✓ should be reverted when msg.sender is not owner
  ✓ should be reverted when rate > 100
  ✓ should change kaiAdmin
setVotingKSPMiningRate()
  ✓ should be reverted when msg.sender is not owner
  ✓ should be reverted when rate > 100
 valid case
   ✓ should set Mining
   ✓ should update vKSPMiningRate
setMaxMiningPoolCount()
  ✓ should be reverted when msg.sender is not owner
  ✓ should be reverted when rate > 100
 valid case
   ✓ should set MAX_MINING_POOL_COUNT
   ✓ should update PoolRanking
sendReward()
  ✓ should be reverted when msg.sender is not votingKSP
 valid case
   ✓ should send Reward
```

```
  setMiningRate()
    ✓ should be reverted when vKSPMiningRate is 0
    ✓ should be reverted when poolCount is 0 (128ms)
    8) should fail when overflow (125ms)
    valid case
      ✓ should change MiningRate

PoolVotingImpl
  version()
    ✓ should return version
  addVoting()
    ✓ should fail when amount is 0
    ✓ should fail when userVotingPoolCount < Max
    valid case : 1st
      ✓ should update userVotingPoolAmount info (54ms)
      ✓ should update poolAmount info (57ms)
      ✓ should update userVotingPoolCount info (55ms)
      ✓ should emit ChangeMiningRate event (46ms)
    valid case : 2nd
      ✓ should update userVotingPoolAmount info
      ✓ should update poolAmount info
  removeVoting()
    ✓ should fail when amount is 0
    ✓ should fail when exchange is 0
    ✓ should fail when not voted
    valid case
      ✓ should update userVotingPoolAmount info (56ms)
      ✓ should update poolAmount info (61ms)
      ✓ should update userVotingPoolCount info (63ms)
      ✓ should emit RemoveVoting event (52ms)
  removeAllVoting()
    ✓ should fail when user is zeroAddress
    valid case
      ✓ should update userVotingPoolAmount info (97ms)
      ✓ should update poolAmount info (96ms)
      ✓ should update userVotingPoolCount info (101ms)
  updatePoolRanking1()
    valid case
      ✓ should update poolRanking info
      ✓ should update poolCount info
  marketUpdateA()
    valid case
```

```
    ✓ should update marketIndexA info
    ✓ should emit UpdateMarketIndex event
  marketUpdateB()
   valid case
    ✓ should update marketIndexA info (42ms)
    ✓ should emit UpdateMarketIndex event
  claimReward()
   valid case
    ✓ should update userRewardSumA info (51ms)
    ✓ should update userRewardSumb info (47ms)
  claimRewardAll()
   valid case
    ✓ should update userRewardSumA info (64ms)
    ✓ should update userRewardSumb info (62ms)

StoreImpl
  version()
    ✓ should return version
  update()
    ✓ should update reserveA (41ms)
    ✓ should update reserveB (47ms)
    ✓ should update blockTimestampLast (41ms)
    ✓ should update priceACumulativeLast (86ms)
    ✓ should update priceBCumulativeLast (84ms)

Store
  constructor()
    ✓ should init implementation
    ✓ should init governance
  _setImplementation()
    ✓ should fail when msg.sender is not governance
    ✓ should fail when imple is equal to newImpl
    ✓ should update implementation

VotingKSPImpl
  version()
    ✓ should return version
  getUserUnlockTime()
   CASE : before unlock
    ✓ should return unlock Time
   CASE : after unlock before lockPeriod
    ✓ should return appropriate Time (46ms)
```

```
  CASE : after unlock after lockPeriod
    ✓ should return appropriate Time (38ms)
getCurrentBalance()
  ✓ should return 0 when index is zero
  ✓ should return appropriate value
getPriorBalance()
  ✓ should fail when block is already passed
  ✓ should return 0 when index is zero
  ✓ should return snapShotBalance when blockNumber is large
  ✓ should return 0 when blockNumber is small (47ms)
  ✓ should return centerbalance when centerBlock = blockNumber (53ms)
  ✓ should return centerbalance when centerBlock < blockNumber (48ms)
  ✓ should return centerbalance when centerBlock > blockNumber (47ms)
claimRerward()
  ✓ should send reward (50ms)
  ✓ should update userRewardSum
  ✓ should emit GiveReward event
setMining()
  ✓ should fail when msg.sender is not governance
  ✓ should fail when _mining > 100
  valid case
    ✓ should update mining
    ✓ should emit UpdateMiningIndex event
    ✓ should emit ChangeMiningRate event
lockKSP()
  ✓ should fail when amount is zero
  ✓ should fail when period is invalid
  valid case
    ✓ should update lockedKSP (54ms)
    9) should update balanceOf (55ms)
    ✓ should update totalSupply (56ms)
    ✓ should update unlockTime (58ms)
    ✓ should update lockPeriod (63ms)
    ✓ should update snapShotCount (61ms)
    ✓ should emit LockKSP event (59ms)
refixBoosting()
  ✓ should fail lockedKSP[msg.sender] is zero
  ✓ should fail when period is invalid
  valid case
    ✓ should update totalSupply (45ms)
    ✓ should emit Transfer event (44ms)
    ✓ should update balanceOf (46ms)
```

✓ should update unlockTime (48ms)
✓ should update lockPeriod (55ms)
✓ should update snapShotCount (48ms)
✓ should emit RefixBoosting event (42ms)
unlockKSP()
✓ should fail unlockTime[msg.sender] and balanceOf[msg.sender] is zero
valid case
✓ should removeAllVoting (80ms)
✓ should transfer token (71ms)
✓ should update totalSupply (76ms)
✓ should update balanceOf (69ms)
✓ should update unlockTime (74ms)
✓ should update lockPeriod (72ms)
✓ should update snapShotCount (67ms)
✓ should emit UnlockKSP event (71ms)

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/					
KSStore.sol	**100**	**100**	**100**	**100**	
KlaytnExchange.impl.sol	**100**	**100**	**100**	**100**	
KlaytnFactory.impl.sol	**100**	**100**	**100**	**100**	
contracts/kai/					
Admin.impl.sol	**100**	**100**	**100**	**100**	
Kai.sol	**100**	**100**	**100**	**100**	
contracts/supporter/					
Supporter.impl.sol	**100**	**100**	**100**	**100**	
Wallet.impl.sol	**100**	**100**	**100**	**100**	
contracts/treasury/	**100**	**100**	**100**	**100**	
Distribution.impl.sol	**100**	**100**	**100**	**100**	
Treasury.impl.sol	**100**	**100**	**100**	**100**	
contracts/vksp/					
Governance.impl.sol	**100**	**100**	**100**	**100**	

PoolVoting.impl.sol	**100**	**100**	**100**	**100**	
Store.impl.sol	**100**	**100**	**100**	**100**	
VotingKSP.impl.sol	**100**	**100**	**100**	**100**	

[Table 1] Test Case Coverage