KNOWNSEC

Degree of secrecy ★ | Project team open

Smart Contract Audit Report

Security status

Safe

★ ★ ★ ★ ★

Principal tester: Knownsec blockchain security team

Version Summary

Content	Date	Version
Editing Document	20210305	V1.0

Report Information

Title	Version	Document Number	Type
KST Smart Contract Audit Report	V1.0		Open to project team

Copyright Notice

Knownsec only issues this report for facts that have occurred or existed before the issuance of this report, and assumes corresponding responsibilities for this. Knownsec is unable to determine the security status of its smart contracts and is not responsible for the facts that will occur or exist in the future. The security audit analysis and other content made in this report are only based on the documents and information provided to us by the information provider as of the time this report is issued. Knownsec's assumption: There is no missing, tampered, deleted or concealed information. If the information provided is missing, tampered with, deleted, concealed or reflected in the actual situation, Knownsec shall not be liable for any losses and adverse effects caused thereby.

Knownsec

Table of Contents

Knownsec

Knownsec

Knownsec

1. Introduction

The effective test time of this report is from Since February 26, 2021 to March 5, 2021 . During this period, the security and standardization of **the smart contract code of the KST** will be audited and used as the statistical basis for the report.

In this audit report, engineers conducted a comprehensive analysis of the common vulnerabilities of smart contracts (Chapter 3). **The smart contract code of the KST** is comprehensively assessed as **SAFE**.

Results of this smart contract security audit： SAFE

Since the testing is under non-production environment, all codes are the latest version. In addition, the testing process is communicated with the relevant engineer, and testing operations are carried out under the controllable operational risk to avoid production during the testing process, such as: Operational risk, code security risk.

Report information of this audit:

Report Number：

Report query address link:

https://attest.im/attestation/searchResult?qurey=

Target information of the KST audit:

Target information	
Token name	KST
Code type	Token code, OKExChain smart contract code
Code language	solidity

Contract documents and hash:

Contract documents	MD5
IERC20.sol	215DB3D2B6A3BDE9297D164F8CBD4402

Knownsec

IKswapCallee.sol	959A9CCE3F383AB2BB2223D806DB2E88
IKswapERC20.sol	0FDE1C027002442CE15FA8DCCE0EA3DB
IKswapFactory.sol	BC07A55ED0FDF0B2CF2F0A3F4B1D8B4A
IKswapPair.sol	082EAA3F8353537F205D0F7A3160060D
IKswapRouter01.sol	454BE0CF6A45818DBA7E8F32AE6D2F0B
IKswapRouter02.sol	E2379DCB210D8F6E6064F0DCD55201D7
IWOKT.sol	9C194449A8643CE70A0F963CAF825AFD
KSTToken.sol	9A3CE769D6A3516C0EDFC9481B3859E7
FixedPoint.sol	2A3B1D277C9F7F59823275D9AE16EFF0
KswapLibrary.sol	B8FC318D3C4FAD9A855E24ACF4D0F4BD
KswapOracleLibrary.sol	0EFD697ECEF27655234C13A4BCECEE30
Math.sol	BAF379A423703AB9C461DF6C94D3223B
SafeMath.sol	851CEFB0A22D2AE9240054CA76941739
TransferHelper.sol	B312DFEF05A1E7FDBADE150348591BD0
UQ112x112.sol	7283321879C861E2DAB8C07E021B5E65
Oracle.sol	309786C4F7BB8031E8D2736F1C688A25
DepositPool.sol	5E41C8B0CA31F435E0C52AD9F09BDA18
LiquidityPool.sol	A4BD6DCD81B43016DB8455804BFCE266
TradingPoolV2.sol	A759F8A7651B08107BEFD50D0CEA4B5C
KswapERC20.sol	D2CB99FA7AE9EB8800924E085F2CC298

Knownsec

KswapFactory.sol	BC2EF73484DF5A772DDEAEF2D4EFC0A0
KswapPair.sol	39A157E7395D481037FA49C7F2FD5C94
KswapRouter.sol	BFBF39777AC79F101539DA1C8B39A42C
TimeLock.sol	53FCE806BDE23340635E0204D6F2525B

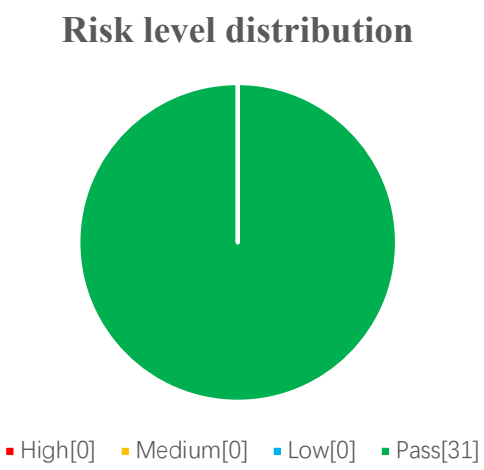
Knownsec

2. Code vulnerability analysis

2.1 Vulnerability Level Distribution

Vulnerability risk statistics by level：

Vulnerability risk level statistics table			
High	Medium	Low	Pass
0	0	0	31



KnownSec

2.2 Audit Result

Result of audit			
Audit Target	**Audit**	**Status**	**Audit Description**
Business security testing	**Oracle price update**	Pass	After testing, there is no such safety vulnerability.
	Pledge pool pledge function	Pass	After testing, there is no such safety vulnerability.
	Liquidity management function	Pass	After testing, there is no such safety vulnerability.
	Functions of the Token Master Contract	Pass	After testing, there is no such safety vulnerability.
Basic code vulnerability detection	**Compiler version security**	Pass	After testing, there is no such safety vulnerability.
	Redundant code	Pass	After testing, there is no such safety vulnerability.
	Use of safe arithmetic library	Pass	After testing, there is no such safety vulnerability.
	Not recommended encoding	Pass	After testing, there is no such safety vulnerability.
	Reasonable use of require/assert	Pass	After testing, there is no such safety vulnerability.
	fallback function safety	Pass	After testing, there is no such safety vulnerability.
	tx.oriigin authentication	Pass	After testing, there is no such safety vulnerability.

	Owner permission control	Pass	After testing, there is no such safety vulnerability.
	Gas consumption detection	Pass	After testing, there is no such safety vulnerability.
	call injection attack	Pass	After testing, there is no such safety vulnerability.
	Low-level function safety	Pass	After testing, there is no such safety vulnerability.
	Vulnerability of additional token issuance	Pass	After testing, there is no such safety vulnerability.
	Access control defect detection	Pass	After testing, there is no such safety vulnerability.
	Numerical overflow detection	Pass	After testing, there is no such safety vulnerability.
	Arithmetic accuracy error	Pass	After testing, there is no such safety vulnerability.
	Wrong use of random number detection	Pass	After testing, there is no such safety vulnerability.
	Unsafe interface use	Pass	After testing, there is no such safety vulnerability.
	Variable coverage	Pass	After testing, there is no such safety vulnerability.
	Uninitialized storage pointer	Pass	After testing, there is no such safety vulnerability.
	Return value call verification	Pass	After testing, there is no such safety vulnerability.

	Transaction order dependency detection	Pass	After testing, there is no such safety vulnerability.
	Timestamp dependent attack	Pass	After testing, there is no such safety vulnerability.
	Denial of service attack detection	Pass	After testing, there is no such safety vulnerability.
	Fake recharge vulnerability detection	Pass	After testing, there is no such safety vulnerability.
	Reentry attack detection	Pass	After testing, there is no such safety vulnerability.
	Replay attack detection	Pass	After testing, there is no such safety vulnerability.
	Rearrangement attack detection	Pass	After testing, there is no such safety vulnerability.

Know

3. Analysis of code audit results

3.1. Oracle price update【PASS】

Audit analysis: The project contract uses Oracle.sol as the oracle price synchronization contract, and uses the update function to update the observation price. After audit, the logic function of the place is normal, and the authority control is correct.

```
function update(address tokenA, address tokenB) external {// knownsec Update Observation Price External Call
    address pair = KswapLibrary.pairFor(factory, tokenA, tokenB);// knownsec Update trading pair

    Observation storage observation = pairObservations[pair];// knownsec Take out the corresponding transaction pair to observe the structure
    uint256 timeElapsed = block.timestamp - observation.timestamp;// knownsec Time calculation
    require(timeElapsed >= CYCLE, "KSWAPOracle: PERIOD_NOT_ELAPSED");// knownsec Update cycle check
    (uint256 price0Cumulative, uint256 price1Cumulative, ) =
        KswapOracleLibrary.currentCumulativePrices(pair);
    observation.timestamp = block.timestamp;// knownsec Time update
    observation.price0Cumulative = price0Cumulative;
    observation.price1Cumulative = price1Cumulative;
}
```

Recommendation： nothing.

Knownsec

3.2. Pledge pool pledge function【PASS】

Audit analysis: The project contract uses DepositPool.sol as the fund pledge pool, deposit as the pledge function, and safeMath to prevent overflow. After audit, the functional design logic of this office is correct, and the authority control is correct.

```
function deposit(uint256 _pid, uint256 _amount) public {// knownsec Pledge
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        updatePool(_pid);// knownsec Pool update
        if (user.amount > 0) {
            uint256 pendingAmount =
                user.amount.mul(pool.accKstPerShare).div(1e12).sub(
                    user.rewardDebt
                );
            if (pendingAmount > 0) {
                safeKstTransfer(msg.sender, pendingAmount);// knownsec Pledge certificate transfer
                user.accKstAmount = user.accKstAmount.add(pendingAmount);
                pool.allocKstAmount = pool.allocKstAmount.sub(pendingAmount);
            }
        }
        if (_amount > 0) {
            ERC20(pool.token).safeTransferFrom(
                msg.sender,
                address(this),
                _amount
            );// knownsec Pledge to transfer currency
            user.amount = user.amount.add(_amount);
            pool.totalAmount = pool.totalAmount.add(_amount);
        }
        user.rewardDebt = user.amount.mul(pool.accKstPerShare).div(1e12);
        emit Deposit(msg.sender, _pid, _amount);
```

Knownsec

```
    }
```

Recommendation： nothing.

3.3. Liquidity management function【PASS】

Audit analysis: The liquidity management function is mainly implemented in KswapRouter.sol. After audit, the function logic in the contract is normal and reasonable, and the authority control is correct.

```
constructor(address _factory, address _WOKT) public {
        factory = _factory;
        WOKT = _WOKT;
    }

    receive() external payable {
        assert(msg.sender == WOKT); // only accept OKT via fallback from the WOKT contract
    }

    function setTradingPool(address _tradingPool) public onlyOwner {
        tradingPool = _tradingPool;
    }

    // **** ADD LIQUIDITY ****
    function _addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin
    ) internal virtual returns (uint256 amountA, uint256 amountB) {
```

Knownsec

```
        // create the pair if it doesn't exist yet
        if (IKswapFactory(factory).getPair(tokenA, tokenB) == address(0)) {
            IKswapFactory(factory).createPair(tokenA, tokenB);
        }
        (uint256 reserveA, uint256 reserveB) =
            KswapLibrary.getReserves(factory, tokenA, tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint256 amountBOptimal =
                KswapLibrary.quote(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(
                    amountBOptimal >= amountBMin,
                    "KswapRouter: INSUFFICIENT_B_AMOUNT"
                );
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint256 amountAOptimal =
                    KswapLibrary.quote(amountBDesired, reserveB, reserveA);
                assert(amountAOptimal <= amountADesired);
                require(
                    amountAOptimal >= amountAMin,
                    "KswapRouter: INSUFFICIENT_A_AMOUNT"
                );
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }

    function addLiquidity(
        address tokenA,
        address tokenB,
```

Knownsec

```
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    )
        external
        virtual
        override
        ensure(deadline)
        returns (
            uint256 amountA,
            uint256 amountB,
            uint256 liquidity
        )
    {
        (amountA, amountB) = _addLiquidity(
            tokenA,
            tokenB,
            amountADesired,
            amountBDesired,
            amountAMin,
            amountBMin
        );
        address pair = KswapLibrary.pairFor(factory, tokenA, tokenB);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = IKswapPair(pair).mint(to);
    }    constructor(address _factory, address _WOKT) public {
        factory = _factory;
        WOKT = _WOKT;
    }
```

Knownsec

```
receive() external payable {
    assert(msg.sender == WOKT); // only accept OKT via fallback from the WOKT contract
}

function setTradingPool(address _tradingPool) public onlyOwner {
    tradingPool = _tradingPool;
}

// **** ADD LIQUIDITY ****
function _addLiquidity(
    address tokenA,
    address tokenB,
    uint256 amountADesired,
    uint256 amountBDesired,
    uint256 amountAMin,
    uint256 amountBMin
) internal virtual returns (uint256 amountA, uint256 amountB) {
    // create the pair if it doesn't exist yet
    if (IKswapFactory(factory).getPair(tokenA, tokenB) == address(0)) {
        IKswapFactory(factory).createPair(tokenA, tokenB);
    }
    (uint256 reserveA, uint256 reserveB) =
        KswapLibrary.getReserves(factory, tokenA, tokenB);
    if (reserveA == 0 && reserveB == 0) {
        (amountA, amountB) = (amountADesired, amountBDesired);
    } else {
        uint256 amountBOptimal =
            KswapLibrary.quote(amountADesired, reserveA, reserveB);
        if (amountBOptimal <= amountBDesired) {
            require(
                amountBOptimal >= amountBMin,
                "KswapRouter: INSUFFICIENT_B_AMOUNT"
```

Knownsec

```
                );
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint256 amountAOptimal =
                    KswapLibrary.quote(amountBDesired, reserveB, reserveA);
                assert(amountAOptimal <= amountADesired);
                require(
                    amountAOptimal >= amountAMin,
                    "KswapRouter: INSUFFICIENT_A_AMOUNT"
                );
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    )
        external
        virtual
        override
        ensure(deadline)
        returns (
            uint256 amountA,
            uint256 amountB,
            uint256 liquidity
```

Knownsec

```
        )
    {
        (amountA, amountB) = _addLiquidity(
            tokenA,
            tokenB,
            amountADesired,
            amountBDesired,
            amountAMin,
            amountBMin
        );
        address pair = KswapLibrary.pairFor(factory, tokenA, tokenB);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = IKswapPair(pair).mint(to);
        }
```

Recommendation： nothing.

3.4. Functions of the Token Master Contract【PASS】

Audit analysis: The main token contract KSTToken.sol, which mainly issues KSTToken tokens, has been audited, and the internal functions of the contract are designed reasonably and the authority control is correct.

```
constructor() public ERC20("KSwap Token", "KST") {
        _mint(msg.sender, preMineSupply);
    }

    // mint with max supply
    function mint(address _to, uint256 _amount)
        public
        onlyMinter
        returns (bool)// knownsec Miners available, minting coins
    {
```

knownsec

```
        if (_amount.add(totalSupply()) > maxSupply) {
            return false;
        }
        _mint(_to, _amount);
        return true;
    }

    function addMinter(address _addMinter) public onlyOwner returns (bool) {// knownsec Owner available, miners increase
        require(
            _addMinter != address(0),
            "KstToken: _addMinter is the zero address"
        );
        return EnumerableSet.add(_minters, _addMinter);
    }

    function delMinter(address _delMinter) public onlyOwner returns (bool) {// knownsec Owner available, miner deleted
        require(
            _delMinter != address(0),
            "KstToken: _delMinter is the zero address"
        );
        return EnumerableSet.remove(_minters, _delMinter);
    }
```

Recommendation： nothing.

knownsec

4. Basic code vulnerability detection

4.1. Compiler version security【PASS】

Check whether a safe compiler version is used in the contract code implementation.

Audit result: After testing, the smart contract code has formulated the compiler version 0.6.0 within the major version, and there is no such security problem.

Recommendation： nothing.

4.2. Redundant code【PASS】

Check whether the contract code implementation contains redundant code.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.3. Use of safe arithmetic library【PASS】

Check whether the SafeMath safe arithmetic library is used in the contract code implementation.

Audit result: After testing, the SafeMath safe arithmetic library has been used in the smart contract code, and there is no such security problem.

Recommendation： nothing.

Knownsec

4.4. Not recommended encoding【PASS】

Check whether there is an encoding method that is not officially recommended or abandoned in the contract code implementation

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.5. Reasonable use of require/assert【PASS】

Check the rationality of the use of require and assert statements in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.6. Fallback function safety【PASS】

Check whether the fallback function is used correctly in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

Knownsec

4.7. tx.origin authentication【PASS】

tx.origin is a global variable of Solidity that traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in a smart contract makes the contract vulnerable to attacks like phishing.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.8. Owner permission control【PASS】

Check whether the owner in the contract code implementation has excessive authority. For example, arbitrarily modify other account balances, etc.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.9. Gas consumption detection【PASS】

Check whether the consumption of gas exceeds the maximum block limit.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

Knownsec

4.10. call injection attack【PASS】

When the call function is called, strict permission control should be done, or the function called by the call should be written dead.

Audit result: After testing, the smart contract does not use the call function, and this vulnerability does not exist.

Recommendation： nothing.

4.11. Low-level function safety【PASS】

Check whether there are security vulnerabilities in the use of low-level functions (call/delegatecall) in the contract code implementation

The execution context of the call function is in the called contract; the execution context of the delegatecall function is in the contract that currently calls the function.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.12. Vulnerability of additional token issuance【PASS】

Check whether there is a function that may increase the total amount of tokens in the token contract after initializing the total amount of tokens.

Knownsec

Audit result: After testing, the smart contract code has the function of issuing additional tokens, but it is only used for the constructor, so it is passed.

Recommendation： nothing.

4.13. Access control defect detection【PASS】

Different functions in the contract should set reasonable permissions.

Check whether each function in the contract correctly uses keywords such as public and private for visibility modification, check whether the contract is correctly defined and use modifier to restrict access to key functions to avoid problems caused by unauthorized access.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.14. Numerical overflow detection【PASS】

The arithmetic problems in smart contracts refer to integer overflow and integer underflow.

Solidity can handle up to 256-bit numbers (2^256-1). If the maximum number increases by 1, it will overflow to 0. Similarly, when the number is an unsigned type, 0 minus 1 will underflow to get the maximum digital value.

Integer overflow and underflow are not a new type of vulnerability, but they are especially dangerous in smart contracts. Overflow conditions can lead to incorrect

Knownsec

results, especially if the possibility is not expected, which may affect the reliability and safety of the program.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.15. Arithmetic accuracy error【PASS】

As a programming language, Solidity has data structure design similar to ordinary programming languages, such as variables, constants, functions, arrays, functions, structures, etc. There is also a big difference between Solidity and ordinary programming languages-Solidity does not float Point type, and all the numerical calculation results of Solidity will only be integers, there will be no decimals, and it is not allowed to define decimal type data. Numerical calculations in the contract are indispensable, and the design of numerical calculations may cause relative errors. For example, the same level of calculations: 5/2*10=20, and 5*10/2=25, resulting in errors, which are larger in data The error will be larger and more obvious.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

Knownsec

4.16. Incorrect use of random numbers【PASS】

Smart contracts may need to use random numbers. Although the functions and variables provided by Solidity can access values that are obviously unpredictable, such as block.number and block.timestamp, they are usually more public than they appear or are affected by miners. These random numbers are predictable to a certain extent, so malicious users can usually copy it and rely on its unpredictability to attack the function.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.17. Unsafe interface usage【PASS】

Check whether unsafe interfaces are used in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.18. Variable coverage【PASS】

Check whether there are security issues caused by variable coverage in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Knownsec

Recommendation： nothing.

4.19. Uninitialized storage pointer【PASS】

In solidity, a special data structure is allowed to be a struct structure, and the local variables in the function are stored in storage or memory by default.

The existence of storage (memory) and memory (memory) are two different concepts. Solidity allows pointers to point to an uninitialized reference, while uninitialized local storage will cause variables to point to other storage variables, leading to variable coverage, or even more serious As a consequence, you should avoid initializing struct variables in functions during development.

Audit result: After testing, the smart contract code does not use structure, there is no such problem.

Recommendation： nothing.

4.20. Return value call verification【PASS】

This problem mostly occurs in smart contracts related to currency transfer, so it is also called silent failed delivery or unchecked delivery.

In Solidity, there are transfer(), send(), call.value() and other currency transfer methods, which can all be used to send OKT to an address. The difference is: When the transfer fails, it will be thrown and the state will be rolled back; Only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when send fails; only 2300gas will be passed for calling to prevent reentry attacks; false will be

Knownsec

returned when call.value fails to be sent; all available gas will be passed for calling (can be Limit by passing in gas_value parameters), which cannot effectively prevent reentry attacks.

If the return value of the above send and call.value transfer functions is not checked in the code, the contract will continue to execute the following code, which may lead to unexpected results due to OKT sending failure.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.21. Transaction order dependency【PASS】

Since miners always get gas fees through codes that represent externally owned addresses (EOA), users can specify higher fees for faster transactions. Since the Ethereum blockchain is public, everyone can see the content of other people's pending transactions. This means that if a user submits a valuable solution, a malicious user can steal the solution and copy its transaction at a higher fee to preempt the original solution.

Audit result: After testing, the _approve function in the contract has a transaction sequence dependency attack risk, but the vulnerability is extremely difficult to exploit, so it is rated as passed. The code is as follows:

```
function _approve(address owner, address spender, uint256 amount) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");
```

Knownsec

```
        _allowances[owner][spender] = amount;//knownnsec// Transaction order depends on
risk        emit Approval(owner, spender, amount);
    }
```

The possible security risks are described as follows:

1. By calling the approve function, user A allows user B to transfer money on his behalf to N (N>0);

2. After a period of time, user A decides to change N to M (M>0), so call the approve function again;

3. User B quickly calls the transferFrom function to transfer N number of tokens before the second call is processed by the miner;

4. After user A's second call to approve is successful, user B can obtain M's transfer quota again, that is, user B obtains N+M's transfer quota through the transaction sequence attack.

Recommendation：

1. Front-end restriction, when user A changes the quota from N to M, he can first change from N to 0, and then from 0 to M.

2. Add the following code at the beginning of the approve function:

require((_value == 0) || (allowed[msg.sender][_spender] == 0));

4.22. Timestamp dependency attack【PASS】

The timestamp of the data block usually uses the local time of the miner, and this time can fluctuate in the range of about 900 seconds. When other nodes accept a new block, it only needs to verify whether the timestamp is later than the previous block

Knownsec

and The error with local time is within 900 seconds. A miner can profit from it by setting the timestamp of the block to satisfy the conditions that are beneficial to him as much as possible.

Check whether there are key functions that depend on the timestamp in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.23. Denial of service attack【PASS】

In the world of Ethereum, denial of service is fatal, and a smart contract that has suffered this type of attack may never be able to return to its normal working state. There may be many reasons for the denial of service of the smart contract, including malicious behavior as the transaction recipient, artificially increasing the gas required for computing functions to cause gas exhaustion, abusing access control to access the private component of the smart contract, using confusion and negligence, etc. Wait.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

Knownsec

4.24. Fake recharge vulnerability【PASS】

The transfer function of the token contract uses the if judgment method to check the balance of the transfer initiator (msg.sender). When balances[msg.sender] <value, enter the else logic part and return false, and finally no exception is thrown. We believe that only if/else this kind of gentle judgment method is an imprecise coding method in sensitive function scenarios such as transfer.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.25. Reentry attack detection【PASS】

The **call.value()** function in Solidity consumes all the gas it receives when it is used to send OKT. When the **call.value()** function to send OKT occurs before the actual reduction of the sender's account balance, There is a risk of reentry attacks.

Audit results： After auditing, the vulnerability does not exist in the smart contract code.

Recommendation： nothing.

4.26. Replay attack detection【PASS】

If the contract involves the need for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks

Knownsec

In the asset management system, there are often cases of entrusted management. The principal assigns assets to the trustee for management, and the principal pays a certain fee to the trustee. This business scenario is also common in smart contracts.

Audit results： After testing, the smart contract does not use the call function, and this vulnerability does not exist.

Recommendation： nothing.

4.27. Rearrangement attack detection【PASS】

A rearrangement attack refers to a miner or other party trying to "compete" with smart contract participants by inserting their own information into a list or mapping, so that the attacker has the opportunity to store their own information in the contract. in.

Audit results： After auditing, the vulnerability does not exist in the smart contract code.

Recommendation： nothing.

Knownsec

5. Appendix A：Contract code

Source code：

IERC20.sol

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0;

interface IERC20Kswap {
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );
    event Transfer(address indexed from, address indexed to, uint256 value);

    function name() external view returns (string memory);

    function symbol() external view returns (string memory);

    function decimals() external view returns (uint8);

    function totalSupply() external view returns (uint256);

    function balanceOf(address owner) external view returns (uint256);

    function allowance(address owner, address spender)
        external
        view
        returns (uint256);

    function approve(address spender, uint256 value) external returns (bool);

    function transfer(address to, uint256 value) external returns (bool);

    function transferFrom(
        address from,
        address to,
        uint256 value
    ) external returns (bool);
}
```

IKswapCallee.sol

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0;

interface IKswapCallee {
    function KswapCall(
        address sender,
        uint256 amount0,
        uint256 amount1,
        bytes calldata data
    ) external;
}
```

IKswapERC20.sol

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0;

interface IKswapERC20 {
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );
    event Transfer(address indexed from, address indexed to, uint256 value);

    function name() external pure returns (string memory);

    function symbol() external pure returns (string memory);

    function decimals() external pure returns (uint8);

    function totalSupply() external view returns (uint256);
```

Knownsec

```
    function balanceOf(address owner) external view returns (uint256);

    function allowance(address owner, address spender)
        external
        view
        returns (uint256);

    function approve(address spender, uint256 value) external returns (bool);

    function transfer(address to, uint256 value) external returns (bool);

    function transferFrom(
        address from,
        address to,
        uint256 value
    ) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);

    function PERMIT_TYPEHASH() external pure returns (bytes32);

    function nonces(address owner) external view returns (uint256);

    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;
}
```

IKswapFactory.sol

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0;

interface IKswapFactory {
    event PairCreated(
        address indexed token0,
        address indexed token1,
        address pair,
        uint256
    );

    function feeTo() external view returns (address);

    function feeToSetter() external view returns (address);

    function feeToRate() external view returns (uint256);

    function getPair(address tokenA, address tokenB)
        external
        view
        returns (address pair);

    function allPairs(uint256) external view returns (address pair);

    function allPairsLength() external view returns (uint256);

    function createPair(address tokenA, address tokenB)
        external
        returns (address pair);

    function setFeeTo(address) external;

    function setFeeToSetter(address) external;

    function setFeeToRate(uint256) external;
}
```

IKswapPair.sol

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0;

interface IKswapPair {
    event Approval(
        address indexed owner,
        address indexed spender,
```

Knownsec

```
        uint256 value
);
event Transfer(address indexed from, address indexed to, uint256 value);

function name() external pure returns (string memory);

function symbol() external pure returns (string memory);

function decimals() external pure returns (uint8);

function totalSupply() external view returns (uint256);

function balanceOf(address owner) external view returns (uint256);

function allowance(address owner, address spender)
        external
        view
        returns (uint256);

function approve(address spender, uint256 value) external returns (bool);

function transfer(address to, uint256 value) external returns (bool);

function transferFrom(
        address from,
        address to,
        uint256 value
) external returns (bool);

function DOMAIN_SEPARATOR() external view returns (bytes32);

function PERMIT_TYPEHASH() external pure returns (bytes32);

function nonces(address owner) external view returns (uint256);

function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
) external;

event Mint(address indexed sender, uint256 amount0, uint256 amount1);
event Burn(
        address indexed sender,
        uint256 amount0,
        uint256 amount1,
        address indexed to
);
event Swap(
        address indexed sender,
        uint256 amount0In,
        uint256 amount1In,
        uint256 amount0Out,
        uint256 amount1Out,
        address indexed to
);
event Sync(uint112 reserve0, uint112 reserve1);

function MINIMUM_LIQUIDITY() external pure returns (uint256);

function factory() external view returns (address);

function token0() external view returns (address);

function token1() external view returns (address);

function getReserves()
        external
        view
        returns (
                uint112 reserve0,
                uint112 reserve1,
                uint32 blockTimestampLast
        );

function price0CumulativeLast() external view returns (uint256);

function price1CumulativeLast() external view returns (uint256);

function kLast() external view returns (uint256);

function mint(address to) external returns (uint256 liquidity);
```

Knownsec

```
    function burn(address to)
        external
        returns (uint256 amount0, uint256 amount1);

    function swap(
        uint256 amount0Out,
        uint256 amount1Out,
        address to,
        bytes calldata data
    ) external;

    function skim(address to) external;

    function sync() external;

    function price(address token, uint256 baseDecimal)
        external
        view
        returns (uint256);

    function initialize(address, address) external;
}
```

IKswapRouter01.sol

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.2;

interface IKswapRouter01 {
    function factory() external pure returns (address);

    function WOKT() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    )
        external
        returns (
            uint256 amountA,
            uint256 amountB,
            uint256 liquidity
        );

    function addLiquidityETH(
        address token,
        uint256 amountTokenDesired,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    )
        external
        payable
        returns (
            uint256 amountToken,
            uint256 amountETH,
            uint256 liquidity
        );

    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountA, uint256 amountB);

    function removeLiquidityETH(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountToken, uint256 amountETH);
```

Knownsec

```
function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
) external returns (uint256 amountA, uint256 amountB);

function removeLiquidityETHWithPermit(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
) external returns (uint256 amountToken, uint256 amountETH);

function swapExactTokensForTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
) external returns (uint256[] memory amounts);

function swapTokensForExactTokens(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
        uint256 deadline
) external returns (uint256[] memory amounts);

function swapExactETHForTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
) external payable returns (uint256[] memory amounts);

function swapTokensForExactETH(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
        uint256 deadline
) external returns (uint256[] memory amounts);

function swapExactTokensForETH(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
) external returns (uint256[] memory amounts);

function swapETHForExactTokens(
        uint256 amountOut,
        address[] calldata path,
        address to,
        uint256 deadline
) external payable returns (uint256[] memory amounts);

function quote(
        uint256 amountA,
        uint256 reserveA,
        uint256 reserveB
) external pure returns (uint256 amountB);

function getAmountOut(
        uint256 amountIn,
        uint256 reserveIn,
        uint256 reserveOut
) external pure returns (uint256 amountOut);

function getAmountIn(
        uint256 amountOut,
```

Knownsec

```
        uint256 reserveIn,
        uint256 reserveOut
    ) external pure returns (uint256 amountIn);

    function getAmountsOut(uint256 amountIn, address[] calldata path)
        external
        view
        returns (uint256[] memory amounts);

    function getAmountsIn(uint256 amountOut, address[] calldata path)
        external
        view
        returns (uint256[] memory amounts);
}
```

IKswapRouter02.sol

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.2;

import "./IKswapRouter01.sol";

interface IKswapRouter02 is IKswapRouter01 {
    function tradingPool() external pure returns (address);

    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountETH);

    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external returns (uint256 amountETH);

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external;

    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable;

    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external;
}
```

IWOKT.sol

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0;

interface IWOKT {
    function deposit() external payable;

    function transfer(address to, uint256 value) external returns (bool);

    function withdraw(uint256) external;
}
```

Knownsec

KSTToken.sol

```
// SPDX-License-Identifier: MIT
pragma solidity =0.6.12;// knownsec  指定编译器版本
pragma experimental ABIEncoderV2;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/EnumerableSet.sol";

abstract contract DelegateERC20 is ERC20 {// knownsec  治理  投票合约
    // A record of each accounts delegate
    mapping(address => address) internal _delegates;

    // A checkpoint for marking number of votes from a given block
    struct Checkpoint {
        uint32 fromBlock;
        uint256 votes;
    }

    // A record of votes checkpoints for each account, by index
    mapping(address => mapping(uint32 => Checkpoint)) public checkpoints;

    // The number of checkpoints for each account
    mapping(address => uint32) public numCheckpoints;

    // The EIP-712 typehash for the contract's domain
    bytes32 public constant DOMAIN_TYPEHASH =
        keccak256(
            "EIP712Domain(string name,uint256 chainId,address verifyingContract)"
        );

    // The EIP-712 typehash for the delegation struct used by the contract
    bytes32 public constant DELEGATION_TYPEHASH =
        keccak256("Delegation(address delegatee,uint256 nonce,uint256 expiry)");

    // A record of states for signing / validating signatures
    mapping(address => uint256) public nonces;

    // support delegates mint
    function _mint(address account, uint256 amount) internal virtual override {
        super._mint(account, amount);

        // add delegates to the minter
        _moveDelegates(address(0), _delegates[account], amount);
    }

    function _transfer(
        address sender,
        address recipient,
        uint256 amount
    ) internal virtual override {
        super._transfer(sender, recipient, amount);
        _moveDelegates(_delegates[sender], _delegates[recipient], amount);
    }

    /**
     * @notice Delegate votes from `msg.sender` to `delegatee`
     * @param delegatee The address to delegate votes to
     */
    function delegate(address delegatee) external {
        return _delegate(msg.sender, delegatee);
    }

    /**
     * @notice Delegates votes from signatory to `delegatee`
     * @param delegatee The address to delegate votes to
     * @param nonce The contract state required to match the signature
     * @param expiry The time at which to expire the signature
     * @param v The recovery byte of the signature
     * @param r Half of the ECDSA signature pair
     * @param s Half of the ECDSA signature pair
     */
    function delegateBySig(
        address delegatee,
        uint256 nonce,
        uint256 expiry,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external {
        bytes32 domainSeparator =
            keccak256(
                abi.encode(
                    DOMAIN_TYPEHASH,
```

Knownsec

```
                keccak256(bytes(name())),
                getChainId(),
                address(this)
            )
        );

    bytes32 structHash =
        keccak256(
            abi.encode(DELEGATION_TYPEHASH, delegatee, nonce, expiry)
        );

    bytes32 digest =
        keccak256(
            abi.encodePacked("\x19\x01", domainSeparator, structHash)
        );

    address signatory = ecrecover(digest, v, r, s);
    require(
        signatory != address(0),
        "KstToken::delegateBySig: invalid signature"
    );
    require(
        nonce == nonces[signatory]++,
        "KstToken::delegateBySig: invalid nonce"
    );
    require(now <= expiry, "KstToken::delegateBySig: signature expired");
    return _delegate(signatory, delegatee);
}

/**
 * @notice Gets the current votes balance for `account`
 * @param account The address to get votes balance
 * @return The number of current votes for `account`
 */
function getCurrentVotes(address account) external view returns (uint256) {
    uint32 nCheckpoints = numCheckpoints[account];
    return
        nCheckpoints > 0 ? checkpoints[account][nCheckpoints - 1].votes : 0;
}

/**
 * @notice Determine the prior number of votes for an account as of a block number
 * @dev Block number must be a finalized block or else this function will revert to prevent misinformation.
 * @param account The address of the account to check
 * @param blockNumber The block number to get the vote balance at
 * @return The number of votes the account had as of the given block
 */
function getPriorVotes(address account, uint256 blockNumber)
    external
    view
    returns (uint256)
{
    require(
        blockNumber < block.number,
        "KstToken::getPriorVotes: not yet determined"
    );

    uint32 nCheckpoints = numCheckpoints[account];
    if (nCheckpoints == 0) {
        return 0;
    }

    // First check most recent balance
    if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
        return checkpoints[account][nCheckpoints - 1].votes;
    }

    // Next check implicit zero balance
    if (checkpoints[account][0].fromBlock > blockNumber) {
        return 0;
    }

    uint32 lower = 0;
    uint32 upper = nCheckpoints - 1;
    while (upper > lower) {
        uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
        Checkpoint memory cp = checkpoints[account][center];
        if (cp.fromBlock == blockNumber) {
            return cp.votes;
        } else if (cp.fromBlock < blockNumber) {
            lower = center;
        } else {
            upper = center - 1;
        }
    }
    return checkpoints[account][lower].votes;
}
```

Knownsec

```
function _delegate(address delegator, address delegatee) internal {
    address currentDelegate = _delegates[delegator];
    uint256 delegatorBalance = balanceOf(delegator); // balance of underlying balances (not scaled);
    _delegates[delegator] = delegatee;

    _moveDelegates(currentDelegate, delegatee, delegatorBalance);

    emit DelegateChanged(delegator, currentDelegate, delegatee);
}

function _moveDelegates(
    address srcRep,
    address dstRep,
    uint256 amount
) internal {
    if (srcRep != dstRep && amount > 0) {
        if (srcRep != address(0)) {
            // decrease old representative
            uint32 srcRepNum = numCheckpoints[srcRep];
            uint256 srcRepOld =
                srcRepNum > 0
                    ? checkpoints[srcRep][srcRepNum - 1].votes
                    : 0;
            uint256 srcRepNew = srcRepOld.sub(amount);
            _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
        }

        if (dstRep != address(0)) {
            // increase new representative
            uint32 dstRepNum = numCheckpoints[dstRep];
            uint256 dstRepOld =
                dstRepNum > 0
                    ? checkpoints[dstRep][dstRepNum - 1].votes
                    : 0;
            uint256 dstRepNew = dstRepOld.add(amount);
            _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
        }
    }
}

function _writeCheckpoint(
    address delegatee,
    uint32 nCheckpoints,
    uint256 oldVotes,
    uint256 newVotes
) internal {
    uint32 blockNumber =
        safe32(
            block.number,
            "KstToken::_writeCheckpoint: block number exceeds 32 bits"
        );

    if (
        nCheckpoints > 0 &&
        checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber
    ) {
        checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
    } else {
        checkpoints[delegatee][nCheckpoints] = Checkpoint(
            blockNumber,
            newVotes
        );
        numCheckpoints[delegatee] = nCheckpoints + 1;
    }

    emit DelegateVotesChanged(delegatee, oldVotes, newVotes);
}

function safe32(uint256 n, string memory errorMessage)
    internal
    pure
    returns (uint32)
{
    require(n < 2**32, errorMessage);
    return uint32(n);
}

function getChainId() internal pure returns (uint256) {
    uint256 chainId;
    assembly {
        chainId := chainid()
    }

    return chainId;
}
```

Knownsec

```
    /// @notice An event thats emitted when an account changes its delegate
    event DelegateChanged(
        address indexed delegator,
        address indexed fromDelegate,
        address indexed toDelegate
    );

    /// @notice An event thats emitted when a delegate account's vote balance changes
    event DelegateVotesChanged(
        address indexed delegate,
        uint256 previousBalance,
        uint256 newBalance
    );
}

contract KSTToken is DelegateERC20, Ownable {// knownsec KSTToken 合约
    uint256 private constant preMineSupply = 100000000 * 1e18;
    uint256 private constant maxSupply = 1000000000 * 1e18; // the total supply

    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _minters;

    constructor() public ERC20("KSwap Token", "KST") {
        _mint(msg.sender, preMineSupply);
    }

    // mint with max supply
    function mint(address _to, uint256 _amount)
        public
        onlyMinter
        returns (bool)// knownsec 矿工可用 铸币
    {
        if (_amount.add(totalSupply()) > maxSupply) {
            return false;
        }
        _mint(_to, _amount);
        return true;
    }

    function addMinter(address _addMinter) public onlyOwner returns (bool) {// knownsec Owner 可用 矿工增加
        require(
            _addMinter != address(0),
            "KstToken: _addMinter is the zero address"
        );
        return EnumerableSet.add(_minters, _addMinter);
    }

    function delMinter(address _delMinter) public onlyOwner returns (bool) {// knownsec Owner 可用 矿工减少
        require(
            _delMinter != address(0),
            "KstToken: _delMinter is the zero address"
        );
        return EnumerableSet.remove(_minters, _delMinter);
    }

    function getMinterLength() public view returns (uint256) {// knownsec 矿工数量获取
        return EnumerableSet.length(_minters);
    }

    function isMinter(address account) public view returns (bool) {// knownsec 矿工资格查询
        return EnumerableSet.contains(_minters, account);
    }

    function getMinter(uint256 _index) public view onlyOwner returns (address) {// knownsec Owner 可用 获取下标矿工
        require(
            _index <= getMinterLength() - 1,
            "KstToken: index out of bounds"
        );
        return EnumerableSet.at(_minters, _index);
    }

    // modifier for mint function
    modifier onlyMinter() {// knownsec 矿工修饰器
        require(isMinter(msg.sender), "caller is not the minter");
        _;
    }
}
```

FixedPoint.sol

```
// SPDX-License-Identifier: MIT

pragma solidity =0.6.12;
```

knownsec

```
library FixedPoint {
    // range: [0, 2**112 - 1]
    // resolution: 1 / 2**112
    struct uq112x112 {
        uint224 _x;
    }

    // range: [0, 2**144 - 1]
    // resolution: 1 / 2**112
    struct uq144x112 {
        uint256 _x;
    }

    uint8 private constant RESOLUTION = 112;

    // encode a uint112 as a UQ112x112
    function encode(uint112 x) internal pure returns (uq112x112 memory) {
        return uq112x112(uint224(x) << RESOLUTION);
    }

    // encodes a uint144 as a UQ144x112
    function encode144(uint144 x) internal pure returns (uq144x112 memory) {
        return uq144x112(uint256(x) << RESOLUTION);
    }

    // divide a UQ112x112 by a uint112, returning a UQ112x112
    function div(uq112x112 memory self, uint112 x)
        internal
        pure
        returns (uq112x112 memory)
    {
        require(x != 0, "FixedPoint: DIV_BY_ZERO");
        return uq112x112(self._x / uint224(x));
    }

    // multiply a UQ112x112 by a uint, returning a UQ144x112
    // reverts on overflow
    function mul(uq112x112 memory self, uint256 y)
        internal
        pure
        returns (uq144x112 memory)
    {
        uint256 z;
        require(
            y == 0 || (z = uint256(self._x) * y) / y == uint256(self._x),
            "FixedPoint: MULTIPLICATION_OVERFLOW"
        );
        return uq144x112(z);
    }

    // returns a UQ112x112 which represents the ratio of the numerator to the denominator
    // equivalent to encode(numerator).div(denominator)
    function fraction(uint112 numerator, uint112 denominator)
        internal
        pure
        returns (uq112x112 memory)
    {
        require(denominator > 0, "FixedPoint: DIV_BY_ZERO");
        return uq112x112((uint224(numerator) << RESOLUTION) / denominator);
    }

    // decode a UQ112x112 into a uint112 by truncating after the radix point
    function decode(uq112x112 memory self) internal pure returns (uint112) {
        return uint112(self._x >> RESOLUTION);
    }

    // decode a UQ144x112 into a uint144 by truncating after the radix point
    function decode144(uq144x112 memory self) internal pure returns (uint144) {
        return uint144(self._x >> RESOLUTION);
    }
}
```

KswapLibrary.sol

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0;

import "../interfaces/IKswapPair.sol";

import "./SafeMath.sol";

library KswapLibrary {
    using SafeMathKswap for uint256;

    // returns sorted token addresses, used to handle return values from pairs sorted in this order
```

Knownsec

```
    function sortTokens(address tokenA, address tokenB)
        internal
        pure
        returns (address token0, address token1)
    {
        require(tokenA != tokenB, "KswapLibrary: IDENTICAL_ADDRESSES");
        (token0, token1) = tokenA < tokenB
            ? (tokenA, tokenB)
            : (tokenB, tokenA);
        require(token0 != address(0), "KswapLibrary: ZERO_ADDRESS");
    }

    // calculates the CREATE2 address for a pair without making any external calls
    function pairFor(
        address factory,
        address tokenA,
        address tokenB
    ) internal pure returns (address pair) {
        (address token0, address token1) = sortTokens(tokenA, tokenB);
        pair = address(
            uint256(
                keccak256(
                    abi.encodePacked(
                        hex"ff",
                        factory,
                        keccak256(abi.encodePacked(token0, token1)),
                        hex"7f08f1b43a5b37be17b2d24d4f2c6b1311e19eedc53cc4528f0e72cdfb5d8d37"    //
init code hash
                    )
                )
            )
        );
    }

    // fetches and sorts the reserves for a pair
    function getReserves(
        address factory,
        address tokenA,
        address tokenB
    ) internal view returns (uint256 reserveA, uint256 reserveB) {
        (address token0, ) = sortTokens(tokenA, tokenB);
        (uint256 reserve0, uint256 reserve1, ) =
            IKswapPair(pairFor(factory, tokenA, tokenB)).getReserves();
        (reserveA, reserveB) = tokenA == token0
            ? (reserve0, reserve1)
            : (reserve1, reserve0);
    }

    // given some amount of an asset and pair reserves, returns an equivalent amount of the other asset
    function quote(
        uint256 amountA,
        uint256 reserveA,
        uint256 reserveB
    ) internal pure returns (uint256 amountB) {
        require(amountA > 0, "KswapLibrary: INSUFFICIENT_AMOUNT");
        require(
            reserveA > 0 && reserveB > 0,
            "KswapLibrary: INSUFFICIENT_LIQUIDITY"
        );
        amountB = amountA.mul(reserveB) / reserveA;
    }

    // given an input amount of an asset and pair reserves, returns the maximum output amount of the other asset
    function getAmountOut(
        uint256 amountIn,
        uint256 reserveIn,
        uint256 reserveOut
    ) internal pure returns (uint256 amountOut) {
        require(amountIn > 0, "KswapLibrary: INSUFFICIENT_INPUT_AMOUNT");
        require(
            reserveIn > 0 && reserveOut > 0,
            "KswapLibrary: INSUFFICIENT_LIQUIDITY"
        );
        uint256 amountInWithFee = amountIn.mul(997);
        uint256 numerator = amountInWithFee.mul(reserveOut);
        uint256 denominator = reserveIn.mul(1000).add(amountInWithFee);
        amountOut = numerator / denominator;
    }

    // given an output amount of an asset and pair reserves, returns a required input amount of the other asset
    function getAmountIn(
        uint256 amountOut,
        uint256 reserveIn,
        uint256 reserveOut
    ) internal pure returns (uint256 amountIn) {
        require(amountOut > 0, "KswapLibrary: INSUFFICIENT_OUTPUT_AMOUNT");
        require(
```

Knownsec

```
                reserveIn > 0 && reserveOut > 0,
                "KswapLibrary: INSUFFICIENT_LIQUIDITY"
            );
            uint256 numerator = reserveIn.mul(amountOut).mul(1000);
            uint256 denominator = reserveOut.sub(amountOut).mul(997);
            amountIn = (numerator / denominator).add(1);
        }

        // performs chained getAmountOut calculations on any number of pairs
        function getAmountsOut(
            address factory,
            uint256 amountIn,
            address[] memory path
        ) internal view returns (uint256[] memory amounts) {
            require(path.length >= 2, "KswapLibrary: INVALID_PATH");
            amounts = new uint256[](path.length);
            amounts[0] = amountIn;
            for (uint256 i; i < path.length - 1; i++) {
                (uint256 reserveIn, uint256 reserveOut) =
                    getReserves(factory, path[i], path[i + 1]);
                amounts[i + 1] = getAmountOut(amounts[i], reserveIn, reserveOut);
            }
        }

        // performs chained getAmountIn calculations on any number of pairs
        function getAmountsIn(
            address factory,
            uint256 amountOut,
            address[] memory path
        ) internal view returns (uint256[] memory amounts) {
            require(path.length >= 2, "KswapLibrary: INVALID_PATH");
            amounts = new uint256[](path.length);
            amounts[amounts.length - 1] = amountOut;
            for (uint256 i = path.length - 1; i > 0; i--) {
                (uint256 reserveIn, uint256 reserveOut) =
                    getReserves(factory, path[i - 1], path[i]);
                amounts[i - 1] = getAmountIn(amounts[i], reserveIn, reserveOut);
            }
        }
    }
```

KswapOracleLibrary.sol

```
    // SPDX-License-Identifier: MIT

    pragma solidity =0.6.12;

    import "./FixedPoint.sol";
    import "../interfaces/IKswapPair.sol";

    library KswapOracleLibrary {
        using FixedPoint for *;

        // helper function that returns the current block timestamp within the range of uint32, i.e. [0, 2**32 - 1]
        function currentBlockTimestamp() internal view returns (uint32) {
            return uint32(block.timestamp % 2**32);
        }

        // produces the cumulative price using counterfactuals to save gas and avoid a call to sync.
        function currentCumulativePrices(address pair)
            internal
            view
            returns (
                uint256 price0Cumulative,
                uint256 price1Cumulative,
                uint32 blockTimestamp
            )
        {
            blockTimestamp = currentBlockTimestamp();
            price0Cumulative = IKswapPair(pair).price0CumulativeLast();
            price1Cumulative = IKswapPair(pair).price1CumulativeLast();

            // if time has elapsed since the last update on the pair, mock the accumulated price values
            (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast) =
                IKswapPair(pair).getReserves();
            if (blockTimestampLast != blockTimestamp) {
                // subtraction overflow is desired
                uint32 timeElapsed = blockTimestamp - blockTimestampLast;
                // addition overflow is desired
                // counterfactual
                price0Cumulative +=
                    uint256(FixedPoint.fraction(reserve1, reserve0)._x) *
                    timeElapsed;
                // counterfactual
                price1Cumulative +=
                    uint256(FixedPoint.fraction(reserve0, reserve1)._x) *
```

Knownsec

```
                timeElapsed;
            }
        }
}
```

Math.sol

```
// SPDX-License-Identifier: MIT

pragma solidity =0.6.12;

// a library for performing various math operations

library Math {
    function min(uint x, uint y) internal pure returns (uint z) {
        z = x < y ? x : y;
    }

    // babylonian method (https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Babylonian_method)
    function sqrt(uint y) internal pure returns (uint z) {
        if (y > 3) {
            z = y;
            uint x = y / 2 + 1;
            while (x < z) {
                z = x;
                x = (y / x + x) / 2;
            }
        } else if (y != 0) {
            z = 1;
        }
    }
}
```

SafeMath.sol

```
// SPDX-License-Identifier: MIT

pragma solidity =0.6.12;

// a library for performing overflow-safe math, courtesy of DappHub (https://github.com/dapphub/ds-math)

library SafeMathKswap {
    function add(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require((z = x + y) >= x, "ds-math-add-overflow");
    }

    function sub(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require((z = x - y) <= x, "ds-math-sub-underflow");
    }

    function mul(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require(y == 0 || (z = x * y) / y == x, "ds-math-mul-overflow");
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    function div(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }
}
```

TransferHelper.sol

```
// SPDX-License-Identifier: GPL-3.0-or-later

pragma solidity >=0.6.0;

// helper methods for interacting with ERC20 tokens and sending ETH that do not consistently return true/false
library TransferHelper {
    function safeApprove(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value));
```

Knownsec

```
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: APPROVE_FAILED');
    }

    function safeTransfer(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: TRANSFER_FAILED');
    }

    function safeTransferFrom(address token, address from, address to, uint value) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: TRANSFER_FROM_FAILED');
    }

    function safeTransferETH(address to, uint value) internal {
        (bool success,) = to.call{value:value}(new bytes(0));
        require(success, 'TransferHelper: ETH_TRANSFER_FAILED');
    }
}
```

UQ112x112.sol

```
// SPDX-License-Identifier: MIT

pragma solidity =0.6.12;

// a library for handling binary fixed point numbers (https://en.wikipedia.org/wiki/Q_(number_format))

// range: [0, 2**112 - 1]
// resolution: 1 / 2**112

library UQ112x112 {
    uint224 constant Q112 = 2**112;

    // encode a uint112 as a UQ112x112
    function encode(uint112 y) internal pure returns (uint224 z) {
        z = uint224(y) * Q112; // never overflows
    }

    // divide a UQ112x112 by a uint112, returning a UQ112x112
    function uqdiv(uint224 x, uint112 y) internal pure returns (uint224 z) {
        z = x / uint224(y);
    }
}
```

Oracle.sol

```
// SPDX-License-Identifier: MIT

pragma solidity =0.6.12;// knownsec 指定编译器版本

import "@openzeppelin/contracts/math/SafeMath.sol";
import "../libraries/FixedPoint.sol";
import "../libraries/KswapOracleLibrary.sol";
import "../libraries/KswapLibrary.sol";
import "../interfaces/IKswapFactory.sol";

contract Oracle {// knownsec 预言机
    using FixedPoint for *;
    using SafeMath for uint256;// knownsec 安全数学库使用声明

    struct Observation {// knownsec 观察结构体
        uint256 timestamp;
        uint256 price0Cumulative;
        uint256 price1Cumulative;
    }

    address public immutable factory;// knownsec 工厂合约地址
    uint256 public constant CYCLE = 30 minutes;

    // mapping from pair address to a list of price observations of that pair
    mapping(address => Observation) public pairObservations;

    constructor(address factory_) public {// knownsec 初始化赋值工厂合约地址
        factory = factory_;
    }

    function update(address tokenA, address tokenB) external {// knownsec 更新观察价格 外部调用
        address pair = KswapLibrary.pairFor(factory, tokenA, tokenB);// knownsec 更新交易对

        Observation storage observation = pairObservations[pair];// knownsec 取出对应交易对观察结构体
```

Knownsec

```
        uint256 timeElapsed = block.timestamp - observation.timestamp;// knownsec 时间计算
        require(timeElapsed >= CYCLE, "KSWAPOracle: PERIOD_NOT_ELAPSED");// knownsec 更新周期检查
        (uint256 price0Cumulative, uint256 price1Cumulative, ) =
            KswapOracleLibrary.currentCumulativePrices(pair);
        observation.timestamp = block.timestamp;// knownsec 时间更新
        observation.price0Cumulative = price0Cumulative;
        observation.price1Cumulative = price1Cumulative;
    }

    function computeAmountOut(
        uint256 priceCumulativeStart,
        uint256 priceCumulativeEnd,
        uint256 timeElapsed,
        uint256 amountIn
    ) private pure returns (uint256 amountOut) {// knownsec 输出价格计算
        // overflow is desired.
        FixedPoint.uq112x112 memory priceAverage =
            FixedPoint.uq112x112(
                uint224(
                    (priceCumulativeEnd - priceCumulativeStart) / timeElapsed
                )
            );// knownsec 平均价格计算
        amountOut = priceAverage.mul(amountIn).decode144();
    }

    function consult(
        address tokenIn,
        uint256 amountIn,
        address tokenOut
    ) external view returns (uint256 amountOut) {// knownsec 外部询价
        address pair = KswapLibrary.pairFor(factory, tokenIn, tokenOut);
        Observation storage observation = pairObservations[pair];
        uint256 timeElapsed = block.timestamp - observation.timestamp;
        (uint256 price0Cumulative, uint256 price1Cumulative, ) =
            KswapOracleLibrary.currentCumulativePrices(pair);
        (address token0, ) = KswapLibrary.sortTokens(tokenIn, tokenOut);

        if (token0 == tokenIn) {
            return
                computeAmountOut(
                    observation.price0Cumulative,
                    price0Cumulative,
                    timeElapsed,
                    amountIn
                );
        } else {
            return
                computeAmountOut(
                    observation.price1Cumulative,
                    price1Cumulative,
                    timeElapsed,
                    amountIn
                );
        }
    }
}
```

DepositPool.sol

```
// SPDX-License-Identifier: MIT
pragma solidity =0.6.12;
pragma experimental ABIEncoderV2;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/EnumerableSet.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "../KstToken.sol";

contract DepositPool is Ownable {// knownsec 质押池
    using SafeMath for uint256;
    using SafeERC20 for ERC20;

    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _tokens;

    // Info of each user.
    struct UserInfo {// knownsec 用户信息
        uint256 amount; // How many LP tokens the user has provided.
        uint256 rewardDebt; // Reward debt.
        uint256 accKstAmount; // How many rewards the user has got.
    }
```

Knownsec

```
    struct UserView {
        uint256 stakedAmount;
        uint256 unclaimedRewards;
        uint256 tokenBalance;
        uint256 accKstAmount;
    }

    // Info of each pool.
    struct PoolInfo {// knownsec 池信息
        address token; // Address of LP token contract.
        uint256 allocPoint; // How many allocation points assigned to this pool. ksts to distribute per block.
        uint256 lastRewardBlock; // Last block number that ksts distribution occurs.
        uint256 accKstPerShare; // Accumulated ksts per share, times 1e12.
        uint256 totalAmount; // Total amount of current pool deposit.
        uint256 allocKstAmount;
        uint256 accKstAmount;
    }

    struct PoolView {
        uint256 pid;
        uint256 allocPoint;
        uint256 lastRewardBlock;
        uint256 rewardsPerBlock;
        uint256 accKstPerShare;
        uint256 allocKstAmount;
        uint256 accKstAmount;
        uint256 totalAmount;
        address token;
        string symbol;
        string name;
        uint8 decimals;
    }

    // The KST Token!
    KSTToken public kst;
    // kst tokens created per block.
    uint256 public kstPerBlock;
    // Info of each pool.
    PoolInfo[] public poolInfo;// knownsec 池信息列表
    // Info of each user that stakes LP tokens.
    mapping(uint256 => mapping(address => UserInfo)) public userInfo;// knownsec 用户信息映射
    // pid corresponding address
    mapping(address => uint256) public tokenOfPid;
    // Total allocation points. Must be the sum of all allocation points in all pools.
    uint256 public totalAllocPoint = 0;
    // The block number when kst mining starts.
    uint256 public startBlock;
    uint256 public halvingPeriod = 2628000; // half year

    event Deposit(address indexed user, uint256 indexed pid, uint256 amount);// knownsec 事件记录
    event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
    event EmergencyWithdraw(
        address indexed user,
        uint256 indexed pid,
        uint256 amount
    );

    constructor(
        KSTToken _kst,
        uint256 _kstPerBlock,
        uint256 _startBlock
    ) public {// knownsec 初始化构造函数
        kst = _kst;
        kstPerBlock = _kstPerBlock;
        startBlock = _startBlock;
    }

    function phase(uint256 blockNumber) public view returns (uint256) {// knownsec 查询周期段
        if (halvingPeriod == 0) {
            return 0;
        }
        if (blockNumber > startBlock) {
            return (blockNumber.sub(startBlock).sub(1)).div(halvingPeriod);
        }
        return 0;
    }

    function getKstPerBlock(uint256 blockNumber) public view returns (uint256) {
        uint256 _phase = phase(blockNumber);
        return kstPerBlock.div(2**_phase);// knownsec 2^ ((blockNumber-startBlock-1) / 2628000) blockNumber
足够大可能溢出
    }

    function getKstBlockReward(uint256 _lastRewardBlock)
        public
        view
        returns (uint256)// knownsec 区块奖励获取
```

knownsec

```
{
        uint256 blockReward = 0;
        uint256 lastRewardPhase = phase(_lastRewardBlock);
        uint256 currentPhase = phase(block.number);
        while (lastRewardPhase < currentPhase) {
                lastRewardPhase++;
                uint256 height = lastRewardPhase.mul(halvingPeriod).add(startBlock);
                blockReward = blockReward.add(
                        (height.sub(_lastRewardBlock)).mul(getKstPerBlock(height))
                );
                _lastRewardBlock = height;
        }
        blockReward = blockReward.add(
                (block.number.sub(_lastRewardBlock)).mul(
                        getKstPerBlock(block.number)
                )
        );
        return blockReward;
}

// Add a new lp to the pool. Can only be called by the owner.
function add(
        uint256 _allocPoint,
        address _token,
        bool _withUpdate
) public onlyOwner {// knownsec owner 可用 添加 lp token
        require(_token != address(0), "_token is the zero address");// knownsec token 检查

        require(
                !EnumerableSet.contains(_tokens, _token),
                "_token is already added to the pool"
        );
        // return EnumerableSet.add(_tokens, _token);
        EnumerableSet.add(_tokens, _token);

        if (_withUpdate) {// knownsec  需要更新
                massUpdatePools();
        }
        uint256 lastRewardBlock =
                block.number > startBlock ? block.number : startBlock;// knownsec  上一个奖励块计算
        totalAllocPoint = totalAllocPoint.add(_allocPoint);// knownsec totalAllocPoint 更新
        poolInfo.push(// knownsec  池信息增加
                PoolInfo({
                        token: _token,
                        allocPoint: _allocPoint,
                        lastRewardBlock: lastRewardBlock,
                        accKstPerShare: 0,
                        totalAmount: 0,
                        allocKstAmount: 0,
                        accKstAmount: 0
                })
        );
        tokenOfPid[_token] = getPoolLength() - 1;// knownsec pid 更新
}

// Update the given pool's kst allocation point. Can only be called by the owner.
function set(
        uint256 _pid,
        uint256 _allocPoint,
        bool _withUpdate
) public onlyOwner {
        if (_withUpdate) {
                massUpdatePools();
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(
                _allocPoint
        );
        poolInfo[_pid].allocPoint = _allocPoint;
}

// Update reward variables for all pools. Be careful of gas spending!
function massUpdatePools() public {// knownsec  更新所有奖励池
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
                updatePool(pid);
        }
}

// Update reward variables of the given pool to be up-to-date.
function updatePool(uint256 _pid) public {// knownsec  根据 pid 更新池
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {
                return;
        }

        uint256 tokenSupply = ERC20(pool.token).balanceOf(address(this));
        if (tokenSupply == 0) {
```

knownsec

```
            pool.lastRewardBlock = block.number;
            return;
        }

        uint256 blockReward = getKstBlockReward(pool.lastRewardBlock);

        if (blockReward <= 0) {
            return;
        }

        uint256 kstReward =
            blockReward.mul(pool.allocPoint).div(totalAllocPoint);

        bool minRet = kst.mint(address(this), kstReward);// knownsec 奖励铸币
        if (minRet) {
            pool.accKstPerShare = pool.accKstPerShare.add(
                kstReward.mul(1e12).div(tokenSupply)
            );
            pool.allocKstAmount = pool.allocKstAmount.add(kstReward);
            pool.accKstAmount = pool.allocKstAmount.add(kstReward);
        }
        pool.lastRewardBlock = block.number;
    }

    function deposit(uint256 _pid, uint256 _amount) public {// knownsec 质押
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        updatePool(_pid);// knownsec 池更新
        if (user.amount > 0) {
            uint256 pendingAmount =
                user.amount.mul(pool.accKstPerShare).div(1e12).sub(
                    user.rewardDebt
                );
            if (pendingAmount > 0) {
                safeKstTransfer(msg.sender, pendingAmount);// knownsec 质押凭证转币
                user.accKstAmount = user.accKstAmount.add(pendingAmount);
                pool.allocKstAmount = pool.allocKstAmount.sub(pendingAmount);
            }
        }
        if (_amount > 0) {
            ERC20(pool.token).safeTransferFrom(
                msg.sender,
                address(this),
                _amount
            );// knownsec 质押转币
            user.amount = user.amount.add(_amount);
            pool.totalAmount = pool.totalAmount.add(_amount);
        }
        user.rewardDebt = user.amount.mul(pool.accKstPerShare).div(1e12);
        emit Deposit(msg.sender, _pid, _amount);
    }

    function pendingKst(uint256 _pid, address _user)
        public
        view
        returns (uint256)
    {// knownsec 还未转移的 kst 计算
        require(
            _pid <= poolInfo.length - 1,
            "TradingPool: Can not find this pool"
        );// knownsec pid 校验
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accKstPerShare = pool.accKstPerShare;
        uint256 tokenSupply = ERC20(pool.token).balanceOf(address(this));
        if (user.amount > 0) {
            if (block.number > pool.lastRewardBlock) {
                uint256 blockReward = getKstBlockReward(pool.lastRewardBlock);
                uint256 kstReward =
                    blockReward.mul(pool.allocPoint).div(totalAllocPoint);
                accKstPerShare = accKstPerShare.add(
                    kstReward.mul(1e12).div(tokenSupply)
                );
                return
                    user.amount.mul(accKstPerShare).div(1e12).sub(
                        user.rewardDebt
                    );
            }
            if (block.number == pool.lastRewardBlock) {
                return
                    user.amount.mul(accKstPerShare).div(1e12).sub(
                        user.rewardDebt
                    );
            }
        }
        return 0;
    }
```

knownsec

```
    function withdraw(uint256 _pid, uint256 _amount) public {// knownsec 账户回撤
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        require(user.amount >= _amount, "withdraw: not good");
        updatePool(_pid);
        uint256 pendingAmount =
            user.amount.mul(pool.accKstPerShare).div(1e12).sub(user.rewardDebt);
        if (pendingAmount > 0) {
            safeKstTransfer(msg.sender, pendingAmount);// knownsec 奖励发送
            user.accKstAmount = user.accKstAmount.add(pendingAmount);
            pool.allocKstAmount = pool.allocKstAmount.sub(pendingAmount);
        }
        if (_amount > 0) {
            user.amount = user.amount.sub(_amount);// knownsec 代币不足将报错
            pool.totalAmount = pool.totalAmount.sub(_amount);
            ERC20(pool.token).safeTransfer(msg.sender, _amount);// knownsec 质押代币发送
        }
        user.rewardDebt = user.amount.mul(pool.accKstPerShare).div(1e12);
        emit Withdraw(msg.sender, _pid, _amount);
    }

    function harvestAll() public {// knownsec 全部池进行利息提取
        for (uint256 i = 0; i < poolInfo.length; i++) {
            withdraw(i, 0);
        }
    }

    function emergencyWithdraw(uint256 _pid) public {// knownsec 指定池不要利息紧急回撤
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        uint256 amount = user.amount;
        user.amount = 0;
        user.rewardDebt = 0;
        ERC20(pool.token).safeTransfer(msg.sender, amount);
        pool.totalAmount = pool.totalAmount.sub(amount);
        emit EmergencyWithdraw(msg.sender, _pid, amount);
    }

    // Safe kst transfer function, just in case if rounding error causes pool to not have enough ksts.
    function safeKstTransfer(address _to, uint256 _amount) internal {// knownsec kst 转账
        uint256 kstBalance = kst.balanceOf(address(this));
        if (_amount > kstBalance) {
            kst.transfer(_to, kstBalance);
        } else {
            kst.transfer(_to, _amount);
        }
    }

    // Set the number of kst produced by each block
    function setKstPerBlock(uint256 _newPerBlock) public onlyOwner {// knownsec Owner 可用 设置块奖励数
量
        massUpdatePools();
        kstPerBlock = _newPerBlock;
    }

    function setHalvingPeriod(uint256 _block) public onlyOwner {// knownsec Owner 可用 设置 halvingPeriod
        halvingPeriod = _block;
    }

    function getTokensLength() public view returns (uint256) {// knownsec 获取 token 数量
        return EnumerableSet.length(_tokens);
    }

    function getTokens(uint256 _index) public view returns (address) {// knownsec 获取指定下标 token
        require(
            _index <= getTokensLength() - 1,
            "LiquidityPool: index out of bounds"
        );
        return EnumerableSet.at(_tokens, _index);
    }

    function getPoolLength() public view returns (uint256) {// knownsec 获取池长度
        return poolInfo.length;
    }

    function getAllPools() external view returns (PoolInfo[] memory) {// knownsec 获取池数组信息
        return poolInfo;
    }

    function getPoolView(uint256 pid) public view returns (PoolView memory) {// knownsec 获取指定池数据
        require(pid < poolInfo.length, "pid out of range");
        PoolInfo memory pool = poolInfo[pid];
        ERC20 token = ERC20(pool.token);
        string memory symbol = token.symbol();
        string memory name = token.name();
        uint8 decimals = token.decimals();
```

Knownsec

```
            uint256 rewardsPerBlock =
                pool.allocPoint.mul(kstPerBlock).div(totalAllocPoint);
            return
                PoolView({
                    pid: pid,
                    allocPoint: pool.allocPoint,
                    lastRewardBlock: pool.lastRewardBlock,
                    accKstPerShare: pool.accKstPerShare,
                    rewardsPerBlock: rewardsPerBlock,
                    allocKstAmount: pool.allocKstAmount,
                    accKstAmount: pool.accKstAmount,
                    totalAmount: pool.totalAmount,
                    token: address(token),
                    symbol: symbol,
                    name: name,
                    decimals: decimals
                });
    }

    function getPoolViewByAddress(address token)
        public
        view
        returns (PoolView memory)// knownsec  获取指定地址池数据
    {
        uint256 pid = tokenOfPid[token];
        return getPoolView(pid);
    }

    function getAllPoolViews() external view returns (PoolView[] memory) {// knownsec  获取所有池数据
        PoolView[] memory views = new PoolView[](poolInfo.length);
        for (uint256 i = 0; i < poolInfo.length; i++) {
            views[i] = getPoolView(i);
        }
        return views;
    }

    function getUserView(address token, address account)
        public
        view
        returns (UserView memory)// knownsec  获取指定池用户数据
    {
        uint256 pid = tokenOfPid[token];
        UserInfo memory user = userInfo[pid][account];
        uint256 unclaimedRewards = pendingKst(pid, account);
        uint256 tokenBalance = ERC20(token).balanceOf(account);
        return
            UserView({
                stakedAmount: user.amount,
                unclaimedRewards: unclaimedRewards,
                tokenBalance: tokenBalance,
                accKstAmount: user.accKstAmount
            });
    }

    function getUserViews(address account)
        external
        view
        returns (UserView[] memory)// knownsec  获取所有池用户数据
    {
        address token;
        UserView[] memory views = new UserView[](poolInfo.length);
        for (uint256 i = 0; i < poolInfo.length; i++) {
            token = address(poolInfo[i].token);
            views[i] = getUserView(token, account);
        }
        return views;
    }
}
```

LiquidityPool.sol

```
// SPDX-License-Identifier: MIT
pragma solidity =0.6.12;
pragma experimental ABIEncoderV2;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/EnumerableSet.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "../KstToken.sol";
import "../interfaces/IKswapPair.sol";

contract LiquidityPool is Ownable {// knownsec  流动性提供池
    using SafeMath for uint256;
    using SafeERC20 for ERC20;
```

knownsec

```
using EnumerableSet for EnumerableSet.AddressSet;
EnumerableSet.AddressSet private _pairs;

// Info of each user.
struct UserInfo {// knownsec 用户信息
        uint256 amount; // How many LP tokens the user has provided.
        uint256 rewardDebt; // Reward debt.
        uint256 accKstAmount; // How many rewards the user has got.
}

struct UserView {
        uint256 stakedAmount;
        uint256 unclaimedRewards;
        uint256 lpBalance;
        uint256 accKstAmount;
}

// Info of each pool.
struct PoolInfo {// knownsec 池信息
        address lpToken; // Address of LP token contract.
        uint256 allocPoint; // How many allocation points assigned to this pool. ksts to distribute per block.
        uint256 lastRewardBlock; // Last block number that ksts distribution occurs.
        uint256 accKstPerShare; // Accumulated ksts per share, times 1e12.
        uint256 totalAmount; // Total amount of current pool deposit.
        uint256 allocKstAmount;
        uint256 accKstAmount;
}

struct PoolView {
        uint256 pid;
        address lpToken;
        uint256 allocPoint;
        uint256 lastRewardBlock;
        uint256 rewardsPerBlock;
        uint256 accKstPerShare;
        uint256 allocKstAmount;
        uint256 accKstAmount;
        uint256 totalAmount;
        address token0;
        string symbol0;
        string name0;
        uint8 decimals0;
        address token1;
        string symbol1;
        string name1;
        uint8 decimals1;
}

// The KST Token!
KSTToken public kst;
// kst tokens created per block.
uint256 public kstPerBlock;
// Info of each pool.
PoolInfo[] public poolInfo;// knownsec 池信息列表
// Info of each user that stakes LP tokens.
mapping(uint256 => mapping(address => UserInfo)) public userInfo;// knownsec 用户信息映射
// pid corresponding address
mapping(address => uint256) public LpOfPid;
// Total allocation points. Must be the sum of all allocation points in all pools.
uint256 public totalAllocPoint = 0;
// The block number when kst mining starts.
uint256 public startBlock;
uint256 public halvingPeriod = 2628000; // half year

event Deposit(address indexed user, uint256 indexed pid, uint256 amount);// knownsec 事件记录
event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
event EmergencyWithdraw(
        address indexed user,
        uint256 indexed pid,
        uint256 amount
);

constructor(
        KSTToken _kst,
        uint256 _kstPerBlock,
        uint256 _startBlock
) public {// knownsec 初始化构造函数
        kst = _kst;
        kstPerBlock = _kstPerBlock;
        startBlock = _startBlock;
}

function phase(uint256 blockNumber) public view returns (uint256) {// knownsec 查询周期段
        if (halvingPeriod == 0) {
                return 0;
        }
```

knownsec

```
        if (blockNumber > startBlock) {
            return (blockNumber.sub(startBlock).sub(1)).div(halvingPeriod);
        }
        return 0;
    }

    function getKstPerBlock(uint256 blockNumber) public view returns (uint256) {
        uint256 _phase = phase(blockNumber);
        return kstPerBlock.div(2**_phase);
    }

    function getKstBlockReward(uint256 _lastRewardBlock)
        public
        view
        returns (uint256)// knownsec  区块奖励获取
    {
        uint256 blockReward = 0;
        uint256 lastRewardPhase = phase(_lastRewardBlock);
        uint256 currentPhase = phase(block.number);
        while (lastRewardPhase < currentPhase) {
            lastRewardPhase++;
            uint256 height = lastRewardPhase.mul(halvingPeriod).add(startBlock);
            blockReward = blockReward.add(
                (height.sub(_lastRewardBlock)).mul(getKstPerBlock(height))
            );
            _lastRewardBlock = height;
        }
        blockReward = blockReward.add(
            (block.number.sub(_lastRewardBlock)).mul(
                getKstPerBlock(block.number)
            )
        );
        return blockReward;
    }

    // Add a new lp to the pool. Can only be called by the owner.
    function add(
        uint256 _allocPoint,
        address _lpToken,
        bool _withUpdate
    ) public onlyOwner {// knownsec owner 可用  添加lp token
        require(_lpToken != address(0), "_lpToken is the zero address");// knownsec token 检查

        require(
            !EnumerableSet.contains(_pairs, _lpToken),
            "_lpToken is already added to the pool"
        );
        // return EnumerableSet.add(_pairs, _lpToken);
        EnumerableSet.add(_pairs, _lpToken);

        if (_withUpdate) {
            massUpdatePools();
        }
        uint256 lastRewardBlock =
            block.number > startBlock ? block.number : startBlock;// knownsec  上一个奖励块计算
        totalAllocPoint = totalAllocPoint.add(_allocPoint);// knownsec totalAllocPoint 更新
        poolInfo.push(// knownsec  池信息增加
            PoolInfo({
                lpToken: _lpToken,
                allocPoint: _allocPoint,
                lastRewardBlock: lastRewardBlock,
                accKstPerShare: 0,
                totalAmount: 0,
                allocKstAmount: 0,
                accKstAmount: 0
            })
        );
        LpOfPid[_lpToken] = getPoolLength() - 1;// knownsec pid 更新
    }

    // Update the given pool's kst allocation point. Can only be called by the owner.
    function set(
        uint256 _pid,
        uint256 _allocPoint,
        bool _withUpdate
    ) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(
            _allocPoint
        );
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    // Update reward variables for all pools. Be careful of gas spending!
    function massUpdatePools() public {// knownsec  更新所有奖励池
```

Knownsec

```
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);
        }
    }

    // Update reward variables of the given pool to be up-to-date.
    function updatePool(uint256 _pid) public {// knownsec 根据 pid 更新池
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {
            return;
        }

        uint256 lpSupply = ERC20(pool.lpToken).balanceOf(address(this));
        if (lpSupply == 0) {
            pool.lastRewardBlock = block.number;
            return;
        }

        uint256 blockReward = getKstBlockReward(pool.lastRewardBlock);

        if (blockReward <= 0) {
            return;
        }

        uint256 kstReward =
            blockReward.mul(pool.allocPoint).div(totalAllocPoint);

        bool minRet = kst.mint(address(this), kstReward);// knownsec 奖励铸币
        if (minRet) {
            pool.accKstPerShare = pool.accKstPerShare.add(
                kstReward.mul(1e12).div(lpSupply)
            );
            pool.allocKstAmount = pool.allocKstAmount.add(kstReward);
            pool.accKstAmount = pool.allocKstAmount.add(kstReward);
        }
        pool.lastRewardBlock = block.number;
    }

    function deposit(uint256 _pid, uint256 _amount) public {// knownsec 质押
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        updatePool(_pid);// knownsec 池更新
        if (user.amount > 0) {
            uint256 pendingAmount =
                user.amount.mul(pool.accKstPerShare).div(1e12).sub(
                    user.rewardDebt
                );
            if (pendingAmount > 0) {
                safeKstTransfer(msg.sender, pendingAmount);// knownsec 质押凭证转币
                user.accKstAmount = user.accKstAmount.add(pendingAmount);
                pool.allocKstAmount = pool.allocKstAmount.sub(pendingAmount);
            }
        }
        if (_amount > 0) {
            ERC20(pool.lpToken).safeTransferFrom(
                msg.sender,
                address(this),
                _amount
            );// knownsec 质押转币
            user.amount = user.amount.add(_amount);
            pool.totalAmount = pool.totalAmount.add(_amount);
        }
        user.rewardDebt = user.amount.mul(pool.accKstPerShare).div(1e12);
        emit Deposit(msg.sender, _pid, _amount);
    }

    function pendingKst(uint256 _pid, address _user)
        public
        view
        returns (uint256)
    {// knownsec 还未转移的 kst 计算
        require(
            _pid <= poolInfo.length - 1,
            "TradingPool: Can not find this pool"
        );// knownsec pid 校验
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accKstPerShare = pool.accKstPerShare;
        uint256 lpSupply = ERC20(pool.lpToken).balanceOf(address(this));
        if (user.amount > 0) {
            if (block.number > pool.lastRewardBlock) {
                uint256 blockReward = getKstBlockReward(pool.lastRewardBlock);
                uint256 kstReward =
                    blockReward.mul(pool.allocPoint).div(totalAllocPoint);
                accKstPerShare = accKstPerShare.add(
                    kstReward.mul(1e12).div(lpSupply)
```

Knownsec

```
                );
                return
                    user.amount.mul(accKstPerShare).div(1e12).sub(
                        user.rewardDebt
                    );
            }
            if (block.number == pool.lastRewardBlock) {
                return
                    user.amount.mul(accKstPerShare).div(1e12).sub(
                        user.rewardDebt
                    );
            }
        }
        return 0;
    }

    function withdraw(uint256 _pid, uint256 _amount) public {// knownsec 账户回撤
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        require(user.amount >= _amount, "withdraw: not good");
        updatePool(_pid);
        uint256 pendingAmount =
            user.amount.mul(pool.accKstPerShare).div(1e12).sub(user.rewardDebt);
        if (pendingAmount > 0) {
            safeKstTransfer(msg.sender, pendingAmount);// knownsec 奖励发送
            user.accKstAmount = user.accKstAmount.add(pendingAmount);
            pool.allocKstAmount = pool.allocKstAmount.sub(pendingAmount);
        }
        if (_amount > 0) {
            user.amount = user.amount.sub(_amount);// knownsec 代币不足将报错
            pool.totalAmount = pool.totalAmount.sub(_amount);
            ERC20(pool.lpToken).safeTransfer(msg.sender, _amount);// knownsec 质押代币发送
        }
        user.rewardDebt = user.amount.mul(pool.accKstPerShare).div(1e12);
        emit Withdraw(msg.sender, _pid, _amount);
    }

    function harvestAll() public {// knownsec 全部池进行利息提取
        for (uint256 i = 0; i < poolInfo.length; i++) {
            withdraw(i, 0);
        }
    }

    function emergencyWithdraw(uint256 _pid) public {// knownsec 指定池不要利息紧急回撤
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        uint256 amount = user.amount;
        user.amount = 0;
        user.rewardDebt = 0;
        ERC20(pool.lpToken).safeTransfer(msg.sender, amount);
        pool.totalAmount = pool.totalAmount.sub(amount);
        emit EmergencyWithdraw(msg.sender, _pid, amount);
    }

    // Safe kst transfer function, just in case if rounding error causes pool to not have enough ksts.
    function safeKstTransfer(address _to, uint256 _amount) internal {// knownsec kst 转账
        uint256 kstBalance = kst.balanceOf(address(this));
        if (_amount > kstBalance) {
            kst.transfer(_to, kstBalance);
        } else {
            kst.transfer(_to, _amount);
        }
    }

    // Set the number of kst produced by each block
    function setKstPerBlock(uint256 _newPerBlock) public onlyOwner {// knownsec Owner 可用 设置块奖励数
量
        massUpdatePools();
        kstPerBlock = _newPerBlock;
    }

    function setHalvingPeriod(uint256 _block) public onlyOwner {// knownsec Owner 可用 设置 halvingPeriod
        halvingPeriod = _block;
    }

    function getPairsLength() public view returns (uint256) {// knownsec 获取 token 数量
        return EnumerableSet.length(_pairs);
    }

    function getPairs(uint256 _index) public view returns (address) {// knownsec 获取指定下标 token
        require(
            _index <= getPairsLength() - 1,
            "LiquidityPool: index out of bounds"
        );
        return EnumerableSet.at(_pairs, _index);
    }
```

knownsec

```
function getPoolLength() public view returns (uint256) {// knownsec 获取池长度
    return poolInfo.length;
}

function getAllPools() external view returns (PoolInfo[] memory) {// knownsec 获取池数组信息
    return poolInfo;
}

function getPoolView(uint256 pid) public view returns (PoolView memory) {// knownsec 获取指定池数据
    require(pid < poolInfo.length, "pid out of range");
    PoolInfo memory pool = poolInfo[pid];
    address lpToken = pool.lpToken;
    ERC20 token0 = ERC20(IKswapPair(lpToken).token0());
    ERC20 token1 = ERC20(IKswapPair(lpToken).token1());
    string memory symbol0 = token0.symbol();
    string memory name0 = token0.name();
    uint8 decimals0 = token0.decimals();
    string memory symbol1 = token1.symbol();
    string memory name1 = token1.name();
    uint8 decimals1 = token1.decimals();
    uint256 rewardsPerBlock =
        pool.allocPoint.mul(kstPerBlock).div(totalAllocPoint);
    return
        PoolView({
            pid: pid,
            lpToken: lpToken,
            allocPoint: pool.allocPoint,
            lastRewardBlock: pool.lastRewardBlock,
            accKstPerShare: pool.accKstPerShare,
            rewardsPerBlock: rewardsPerBlock,
            allocKstAmount: pool.allocKstAmount,
            accKstAmount: pool.accKstAmount,
            totalAmount: pool.totalAmount,
            token0: address(token0),
            symbol0: symbol0,
            name0: name0,
            decimals0: decimals0,
            token1: address(token1),
            symbol1: symbol1,
            name1: name1,
            decimals1: decimals1
        });
}

function getPoolViewByAddress(address lpToken)
    public
    view
    returns (PoolView memory)// knownsec 获取指定地址池数据
{
    uint256 pid = LpOfPid[lpToken];
    return getPoolView(pid);
}

function getAllPoolViews() external view returns (PoolView[] memory) {// knownsec 获取所有池数据
    PoolView[] memory views = new PoolView[](poolInfo.length);
    for (uint256 i = 0; i < poolInfo.length; i++) {
        views[i] = getPoolView(i);
    }
    return views;
}

function getUserView(address lpToken, address account)
    public
    view
    returns (UserView memory)// knownsec 获取指定池用户数据
{
    uint256 pid = LpOfPid[lpToken];
    UserInfo memory user = userInfo[pid][account];
    uint256 unclaimedRewards = pendingKst(pid, account);
    uint256 lpBalance = ERC20(lpToken).balanceOf(account);
    return
        UserView({
            stakedAmount: user.amount,
            unclaimedRewards: unclaimedRewards,
            lpBalance: lpBalance,
            accKstAmount: user.accKstAmount
        });
}

function getUserViews(address account)
    external
    view
    returns (UserView[] memory)// knownsec 获取所有池用户数据
{
    address lpToken;
    UserView[] memory views = new UserView[](poolInfo.length);
    for (uint256 i = 0; i < poolInfo.length; i++) {
```

Knownsec

```
                lpToken = address(poolInfo[i].lpToken);
                views[i] = getUserView(lpToken, account);
            }
            return views;
        }
}
```

TradingPoolV2.sol

```
// SPDX-License-Identifier: MIT
pragma solidity =0.6.12;
pragma experimental ABIEncoderV2;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/EnumerableSet.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "../KstToken.sol";
import "../interfaces/IKswapPair.sol";
import "../interfaces/IKswapFactory.sol";
import "../libraries/KswapLibrary.sol";

interface IOracle {
    function update(address tokenA, address tokenB) external;

    function consult(
        address tokenIn,
        uint256 amountIn,
        address tokenOut
    ) external view returns (uint256 amountOut);
}

contract TradingPool is Ownable {// knownsec  交易池
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _pairs;

    // Info of each user.
    struct UserInfo {// knownsec  用户信息
        uint256 quantity;
        uint256 accQuantity;
        uint256 pendingReward;
        uint256 rewardDebt; // Reward debt.
        uint256 accKstAmount; // How many rewards the user has got.
    }

    struct UserView {
        uint256 quantity;
        uint256 accQuantity;
        uint256 unclaimedRewards;
        uint256 accKstAmount;
    }

    // Info of each pool.
    struct PoolInfo {// knownsec  池信息
        address pair; // Address of LP token contract.
        uint256 allocPoint; // How many allocation points assigned to this pool. ksts to distribute per block.
        uint256 lastRewardBlock; // Last block number that ksts distribution occurs.
        uint256 accKstPerShare; // Accumulated ksts per share, times 1e12.
        uint256 quantity;
        uint256 accQuantity;
        uint256 allocKstAmount;
        uint256 accKstAmount;
    }

    struct PoolView {
        uint256 pid;
        address pair;
        uint256 allocPoint;
        uint256 lastRewardBlock;
        uint256 rewardsPerBlock;
        uint256 accKstPerShare;
        uint256 allocKstAmount;
        uint256 accKstAmount;
        uint256 quantity;
        uint256 accQuantity;
        address token0;
        string symbol0;
        string name0;
        uint8 decimals0;
        address token1;
        string symbol1;
```

Knownsec

```
        string name1;
        uint8 decimals1;
    }

    // The KST Token!
    KSTToken public kst;
    // kst tokens created per block.
    uint256 public kstPerBlock;
    // Info of each pool.
    PoolInfo[] public poolInfo;// knownsec 池信息列表
    // Info of each user that stakes LP tokens.
    mapping(uint256 => mapping(address => UserInfo)) public userInfo;// knownsec 用户信息映射
    // pid corresponding address
    mapping(address => uint256) public pairOfPid;
    // Total allocation points. Must be the sum of all allocation points in all pools.
    uint256 public totalAllocPoint = 0;
    uint256 public totalQuantity = 0;
    IOracle public oracle;
    // router address
    address public router;
    // factory address
    IKswapFactory public factory;
    address public targetToken;
    // The block number when kst mining starts.
    uint256 public startBlock;
    uint256 public halvingPeriod = 2628000; // half year

    event Swap(address indexed user, uint256 indexed pid, uint256 amount);// knownsec 事件记录
    event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
    event EmergencyWithdraw(
        address indexed user,
        uint256 indexed pid,
        uint256 amount
    );

    constructor(
        KSTToken _kst,
        IKswapFactory _factory,
        IOracle _oracle,
        address _router,
        address _targetToken,
        uint256 _kstPerBlock,
        uint256 _startBlock
    ) public {// knownsec 初始化构造函数
        kst = _kst;
        factory = _factory;
        oracle = _oracle;
        router = _router;
        targetToken = _targetToken;
        kstPerBlock = _kstPerBlock;
        startBlock = _startBlock;
    }

    function phase(uint256 blockNumber) public view returns (uint256) {// knownsec 查询周期段
        if (halvingPeriod == 0) {
            return 0;
        }
        if (blockNumber > startBlock) {
            return (blockNumber.sub(startBlock).sub(1)).div(halvingPeriod);
        }
        return 0;
    }

    function getKstPerBlock(uint256 blockNumber) public view returns (uint256) {
        uint256 _phase = phase(blockNumber);
        return kstPerBlock.div(2**_phase);
    }

    function getKstBlockReward(uint256 _lastRewardBlock)
        public
        view
        returns (uint256)// knownsec 区块奖励获取
    {
        uint256 blockReward = 0;
        uint256 lastRewardPhase = phase(_lastRewardBlock);
        uint256 currentPhase = phase(block.number);
        while (lastRewardPhase < currentPhase) {
            lastRewardPhase++;
            uint256 height = lastRewardPhase.mul(halvingPeriod).add(startBlock);
            blockReward = blockReward.add(
                (height.sub(_lastRewardBlock)).mul(getKstPerBlock(height))
            );
            _lastRewardBlock = height;
        }
        blockReward = blockReward.add(
            (block.number.sub(_lastRewardBlock)).mul(
                getKstPerBlock(block.number)
```

Knownsec

```
            )
        );
        return blockReward;
    }

    // Only tokens in the whitelist can mine KST
    function addWhitelist(address _addToken) public onlyOwner returns (bool) {// knownsec owner 可用 白名单添加
        require(
            _addToken != address(0),
            "TradingPool: token is the zero address"
        );// knownsec token 检查
        return EnumerableSet.add(_pairs, _addToken);
    }

    function delWhitelist(address _delToken) public onlyOwner returns (bool) {// knownsec owner 可用 白名单删除
        require(
            _delToken != address(0),
            "TradingPool: token is the zero address"
        );
        return EnumerableSet.remove(_pairs, _delToken);
    }

    function getWhitelistLength() public view returns (uint256) {// knownsec 白名单长度获取
        return EnumerableSet.length(_pairs);
    }

    function isWhitelist(address _token) public view returns (bool) {// knownsec 白名单资格查询
        return EnumerableSet.contains(_pairs, _token);
    }

    function getWhitelist(uint256 _index) public view returns (address) {// knownsec 白名单列表获取
        require(
            _index <= getWhitelistLength() - 1,
            "TradingPool: index out of bounds"
        );
        return EnumerableSet.at(_pairs, _index);
    }

    // Add a new pair to the pool. Can only be called by the owner.
    function add(
        uint256 _allocPoint,
        address _pair,
        bool _withUpdate
    ) public onlyOwner {// knownsec owner 可用 添加配对合约
        require(_pair != address(0), "_pair is the zero address");// knownsec token 检查

        require(
            !EnumerableSet.contains(_pairs, _pair),
            "_pair is already added to the pool"
        );
        // return EnumerableSet.add(_pairs, _pair);
        EnumerableSet.add(_pairs, _pair);

        if (_withUpdate) {// knownsec 需要更新
            massUpdatePools();
        }
        uint256 lastRewardBlock =
            block.number > startBlock ? block.number : startBlock;// knownsec 上一个奖励块计算
        totalAllocPoint = totalAllocPoint.add(_allocPoint);// knownsec totalAllocPoint 更新
        poolInfo.push(// knownsec 池信息增加
            PoolInfo({
                pair: _pair,
                allocPoint: _allocPoint,
                lastRewardBlock: lastRewardBlock,
                accKstPerShare: 0,
                quantity: 0,
                accQuantity: 0,
                allocKstAmount: 0,
                accKstAmount: 0
            })
        );
        pairOfPid[_pair] = getPoolLength() - 1;// knownsec pid 更新
    }

    // Update the given pool's kst allocation point. Can only be called by the owner.
    function set(
        uint256 _pid,
        uint256 _allocPoint,
        bool _withUpdate
    ) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(
            _allocPoint
```

Knownsec

```
        );
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    // Update reward variables for all pools. Be careful of gas spending!
    function massUpdatePools() public {// knownsec  更新所有奖励池
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);
        }
    }

    // Update reward variables of the given pool to be up-to-date.
    function updatePool(uint256 _pid) public {// knownsec  根据 pid 更新池
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {
            return;
        }

        if (pool.quantity == 0) {
            pool.lastRewardBlock = block.number;
            return;
        }

        uint256 blockReward = getKstBlockReward(pool.lastRewardBlock);

        if (blockReward <= 0) {
            return;
        }

        uint256 kstReward =
            blockReward.mul(pool.allocPoint).div(totalAllocPoint);

        bool minRet = kst.mint(address(this), kstReward);// knownsec  奖励铸币
        if (minRet) {
            pool.accKstPerShare = pool.accKstPerShare.add(
                kstReward.mul(1e12).div(pool.quantity)
            );
            pool.allocKstAmount = pool.allocKstAmount.add(kstReward);
            pool.accKstAmount = pool.allocKstAmount.add(kstReward);
        }
        pool.lastRewardBlock = block.number;
    }

    function swap(// knownsec  换币
        address account,
        address input,
        address output,
        uint256 amount
    ) public onlyRouter returns (bool) {// knownsec  路由合约可用
        require(
            account != address(0),// knownsec  输入检查
            "TradingPool: swap account is zero address"
        );
        require(input != address(0), "TradingPool: swap input is zero address");
        require(
            output != address(0),
            "TradingPool: swap output is zero address"
        );

        if (getPoolLength() <= 0) {
            return false;
        }

        if (!isWhitelist(input) || !isWhitelist(output)) {
            return false;
        }

        address pair = KswapLibrary.pairFor(address(factory), input, output);// knownsec  获取交易对

        PoolInfo storage pool = poolInfo[pairOfPid[pair]];// knownsec  获取池对象
        // If it does not exist or the allocPoint is 0 then return
        if (pool.pair != pair || pool.allocPoint <= 0) {
            return false;
        }

        uint256 quantity = getQuantity(output, amount, targetToken);// knownsec  转换量计算
        if (quantity <= 0) {
            return false;
        }

        updatePool(pairOfPid[pair]);

        UserInfo storage user = userInfo[pairOfPid[pair]][account];// knownsec  用户对象获取
        if (user.quantity > 0) {
            uint256 pendingReward =
                user.quantity.mul(pool.accKstPerShare).div(1e12).sub(
```

knownsec

```
                    user.rewardDebt
                );// knownsec  用户奖励计算
            if (pendingReward > 0) {
                user.pendingReward = pendingReward;
            }
        }

        if (quantity > 0) {// knownsec swap
            pool.quantity = pool.quantity.add(quantity);
            pool.accQuantity = pool.accQuantity.add(quantity);
            totalQuantity = totalQuantity.add(quantity);
            user.quantity = user.quantity.add(quantity);
            user.accQuantity = user.accQuantity.add(quantity);
        }
        user.rewardDebt = user.quantity.mul(pool.accKstPerShare).div(1e12);
        emit Swap(account, pairOfPid[pair], quantity);// knownsec  时间记录

        return true;
    }

    function pendingKst(uint256 _pid, address _user)
        public
        view
        returns (uint256)
    {// knownsec  还未转移的 kst 计算
        require(
            _pid <= poolInfo.length - 1,
            "TradingPool: Can not find this pool"
        );

        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accKstPerShare = pool.accKstPerShare;

        if (user.quantity > 0) {
            if (block.number > pool.lastRewardBlock) {
                uint256 blockReward = getKstBlockReward(pool.lastRewardBlock);
                uint256 kstReward =
                    blockReward.mul(pool.allocPoint).div(totalAllocPoint);
                accKstPerShare = accKstPerShare.add(
                    kstReward.mul(1e12).div(pool.quantity)
                );
                return
                    user.pendingReward.add(
                        user.quantity.mul(accKstPerShare).div(1e12).sub(
                            user.rewardDebt
                        )
                    );
            }
            if (block.number == pool.lastRewardBlock) {
                return
                    user.pendingReward.add(
                        user.quantity.mul(accKstPerShare).div(1e12).sub(
                            user.rewardDebt
                        )
                    );
            }
        }
        return 0;
    }

    function withdraw(uint256 _pid) public {// knownsec  账户回撤
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];

        updatePool(_pid);
        uint256 pendingAmount = pendingKst(_pid, msg.sender);

        if (pendingAmount > 0) {
            safeKstTransfer(msg.sender, pendingAmount);// knownsec  奖励发送
            pool.quantity = pool.quantity.sub(user.quantity);
            pool.allocKstAmount = pool.allocKstAmount.sub(pendingAmount);
            user.accKstAmount = user.accKstAmount.add(pendingAmount);
            user.quantity = 0;
            user.rewardDebt = 0;
            user.accKstAmount = user.accKstAmount.add(pendingAmount);
        }
        emit Withdraw(msg.sender, _pid, pendingAmount);
    }

    function harvestAll() public {// knownsec  全部池进行利息提取
        for (uint256 i = 0; i < poolInfo.length; i++) {
            withdraw(i);
        }
    }

    function emergencyWithdraw(uint256 _pid) public {// knownsec  指定池不要利息紧急回撤
```

Knownsec

```
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        safeKstTransfer(msg.sender, user.pendingReward);

        pool.quantity = pool.quantity.sub(user.quantity);
        pool.allocKstAmount = pool.allocKstAmount.sub(user.pendingReward);
        user.accKstAmount = user.accKstAmount.add(user.pendingReward);
        user.quantity = 0;
        user.rewardDebt = 0;
        user.accKstAmount = user.accKstAmount.add(user.pendingReward);
        emit EmergencyWithdraw(msg.sender, _pid, user.quantity);
    }

    // Safe kst transfer function, just in case if rounding error causes pool to not have enough ksts.
    function safeKstTransfer(address _to, uint256 _amount) internal {// knownsec kst 转账
        uint256 kstBalance = kst.balanceOf(address(this));
        if (_amount > kstBalance) {
            kst.transfer(_to, kstBalance);
        } else {
            kst.transfer(_to, _amount);
        }
    }

    // Set the number of kst produced by each block
    function setKstPerBlock(uint256 _newPerBlock) public onlyOwner {// knownsec Owner 可用 设置块奖励数量
        massUpdatePools();
        kstPerBlock = _newPerBlock;
    }

    function setHalvingPeriod(uint256 _block) public onlyOwner {// knownsec Owner 可用 设置 halvingPeriod
        halvingPeriod = _block;
    }

    function setRouter(address newRouter) public onlyOwner {// knownsec Owner 可用 设置路由合约
        require(
            newRouter != address(0),
            "TradingPool: new router is the zero address"
        );
        router = newRouter;
    }

    function setOracle(IOracle _oracle) public onlyOwner {// knownsec Owner 可用 设置预言机地址
        require(
            address(_oracle) != address(0),
            "TradingPool: new oracle is the zero address"
        );
        oracle = _oracle;
    }

    function getPairsLength() public view returns (uint256) {// knownsec 获取 配对合约长度
        return EnumerableSet.length(_pairs);
    }

    function getPairs(uint256 _index) public view returns (address) {// knownsec 获取指定下标配对合约
        require(
            _index <= getPairsLength() - 1,
            "LiquidityPool: index out of bounds"
        );
        return EnumerableSet.at(_pairs, _index);
    }

    function getPoolLength() public view returns (uint256) {// knownsec 获取池长度
        return poolInfo.length;
    }

    function getAllPools() external view returns (PoolInfo[] memory) {// knownsec 获取池信息列表
        return poolInfo;
    }

    function getPoolView(uint256 pid) public view returns (PoolView memory) {// knownsec 获取指定 pid 池信息
        require(pid < poolInfo.length, "pid out of range");
        PoolInfo memory pool = poolInfo[pid];
        address pair = address(pool.pair);
        ERC20 token0 = ERC20(IKswapPair(pair).token0());
        ERC20 token1 = ERC20(IKswapPair(pair).token1());
        string memory symbol0 = token0.symbol();
        string memory name0 = token0.name();
        uint8 decimals0 = token0.decimals();
        string memory symbol1 = token1.symbol();
        string memory name1 = token1.name();
        uint8 decimals1 = token1.decimals();
        uint256 rewardsPerBlock =
            pool.allocPoint.mul(kstPerBlock).div(totalAllocPoint);
        return
            PoolView({
```

knownsec

```
                pid: pid,
                pair: pair,
                allocPoint: pool.allocPoint,
                lastRewardBlock: pool.lastRewardBlock,
                accKstPerShare: pool.accKstPerShare,
                rewardsPerBlock: rewardsPerBlock,
                allocKstAmount: pool.allocKstAmount,
                accKstAmount: pool.accKstAmount,
                quantity: pool.quantity,
                accQuantity: pool.accQuantity,
                token0: address(token0),
                symbol0: symbol0,
                name0: name0,
                decimals0: decimals0,
                token1: address(token1),
                symbol1: symbol1,
                name1: name1,
                decimals1: decimals1
            });
    }

    function getPoolViewByAddress(address pair)
        public
        view
        returns (PoolView memory)
    {
        uint256 pid = pairOfPid[pair];
        return getPoolView(pid);
    }

    function getAllPoolViews() external view returns (PoolView[] memory) {
        PoolView[] memory views = new PoolView[](poolInfo.length);
        for (uint256 i = 0; i < poolInfo.length; i++) {
            views[i] = getPoolView(i);
        }
        return views;
    }

    function getUserView(address pair, address account)
        public
        view
        returns (UserView memory)
    {
        uint256 pid = pairOfPid[pair];
        UserInfo memory user = userInfo[pid][account];
        uint256 unclaimedRewards = pendingKst(pid, account);
        return
            UserView({
                quantity: user.quantity,
                accQuantity: user.accQuantity,
                unclaimedRewards: unclaimedRewards,
                accKstAmount: user.accKstAmount
            });
    }

    function getUserViews(address account)
        external
        view
        returns (UserView[] memory)
    {
        address pair;
        UserView[] memory views = new UserView[](poolInfo.length);
        for (uint256 i = 0; i < poolInfo.length; i++) {
            pair = address(poolInfo[i].pair);
            views[i] = getUserView(pair, account);
        }
        return views;
    }

    modifier onlyRouter() {
        require(msg.sender == router, "TradingPool: caller is not the router");
        _;
    }

    function getQuantity(
        address outputToken,
        uint256 outputAmount,
        address anchorToken
    ) public view returns (uint256) {
        uint256 quantity = 0;
        if (outputToken == anchorToken) {
            quantity = outputAmount;
        } else if (
            IKswapFactory(factory).getPair(outputToken, anchorToken) !=
            address(0)
        ) {
            quantity = IOracle(oracle).consult(
```

knownsec

```
                    outputToken,
                    outputAmount,
                    anchorToken
                );
            } else {
                uint256 length = getWhitelistLength();
                for (uint256 index = 0; index < length; index++) {
                    address intermediate = getWhitelist(index);
                    if (
                        IKswapFactory(factory).getPair(outputToken, intermediate) !=
                        address(0) &&
                        IKswapFactory(factory).getPair(intermediate, anchorToken) !=
                        address(0)
                    ) {
                        uint256 interQuantity =
                            IOracle(oracle).consult(
                                outputToken,
                                outputAmount,
                                intermediate
                            );
                        quantity = IOracle(oracle).consult(
                            intermediate,
                            interQuantity,
                            anchorToken
                        );
                        break;
                    }
                }
            }
            return quantity;
        }
}
```

KswapERC20.sol

```
// SPDX-License-Identifier: MIT

pragma solidity =0.6.12;// knownsec  指定编译器版本

import "../libraries/SafeMath.sol";

contract KswapERC20 {
    using SafeMathKswap for uint256;

    string public constant name = "KSwap LP Token";
    string public constant symbol = "KLP";
    uint8 public constant decimals = 18;
    uint256 public totalSupply;
    mapping(address => uint256) public balanceOf;// knownse  账户余额
    mapping(address => mapping(address => uint256)) public allowance;// knownsec  审批额度

    bytes32 public DOMAIN_SEPARATOR;
    // keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)");
    bytes32 public constant PERMIT_TYPEHASH =
        0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64845d6126c9;
    mapping(address => uint256) public nonces;

    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );
    event Transfer(address indexed from, address indexed to, uint256 value);

    constructor() public {
        uint256 chainId;
        assembly {
            chainId := chainid()// knownsec  利用汇编获取当前链 id
        }
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256(
                    "EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"
                ),
                keccak256(bytes(name)),
                keccak256(bytes("1")),
                chainId,
                address(this)
            )
        );
    }

    function _mint(address to, uint256 value) internal {// knownsec  铸币函数  内部使用
        totalSupply = totalSupply.add(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(address(0), to, value);
```

Knownsec

```
    }

    function _burn(address from, uint256 value) internal {// knownsec 烧币函数 内部使用
        balanceOf[from] = balanceOf[from].sub(value);
        totalSupply = totalSupply.sub(value);
        emit Transfer(from, address(0), value);
    }

    function _approve(
        address owner,
        address spender,
        uint256 value
    ) private {// knownsec 授权函数 私有
        allowance[owner][spender] = value;
        emit Approval(owner, spender, value);
    }

    function _transfer(
        address from,
        address to,
        uint256 value
    ) private {// knownsec 转账 私有
        balanceOf[from] = balanceOf[from].sub(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(from, to, value);
    }

    function approve(address spender, uint256 value) external returns (bool) {// knownsec 外部调用 授权
        _approve(msg.sender, spender, value);
        return true;
    }

    function transfer(address to, uint256 value) external returns (bool) {// knownsec 外部调用 转账
        _transfer(msg.sender, to, value);
        return true;
    }

    function transferFrom(
        address from,
        address to,
        uint256 value
    ) external returns (bool) {// knownsec 外部调用 利用授权转账
        if (allowance[from][msg.sender] != uint256(-1)) {
            allowance[from][msg.sender] = allowance[from][msg.sender].sub(
                value
            );
        }
        _transfer(from, to, value);
        return true;
    }

    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external {
        require(deadline >= block.timestamp, "Kswap: EXPIRED");
        bytes32 digest =
            keccak256(
                abi.encodePacked(
                    "\x19\x01",
                    DOMAIN_SEPARATOR,
                    keccak256(
                        abi.encode(
                            PERMIT_TYPEHASH,
                            owner,
                            spender,
                            value,
                            nonces[owner]++,
                            deadline
                        )
                    )
                )
            );
        address recoveredAddress = ecrecover(digest, v, r, s);
        require(
            recoveredAddress != address(0) && recoveredAddress == owner,
            "Kswap: INVALID_SIGNATURE"
        );
        _approve(owner, spender, value);
    }
}
```

knownsec

KswapFactory.sol

```
// SPDX-License-Identifier: MIT

pragma solidity =0.6.12;// knownsec 指定编译器版本

import "../interfaces/IKswapFactory.sol";
import "../libraries/SafeMath.sol";
import "./KswapPair.sol";

contract KswapFactory is IKswapFactory {// knownsec 工厂合约
    using SafeMathKswap for uint256;

    address public override feeTo;
    address public override feeToSetter;
    uint256 public override feeToRate;

    mapping(address => mapping(address => address)) public override getPair;
    address[] public override allPairs;

    event PairCreated(
        address indexed token0,
        address indexed
        ,
        address pair,
        uint256
    );

    constructor(address _feeToSetter) public {
        feeToSetter = _feeToSetter;// knownsec 初始化设置手续费地址
    }

    function allPairsLength() external view override returns (uint256) {
        return allPairs.length;
    }

    function pairCodeHash() external pure returns (bytes32) {
        return keccak256(type(KswapPair).creationCode);
    }

    function createPair(address tokenA, address tokenB)
        external
        override
        returns (address pair)// knownsec 创建交易对
    {
        require(tokenA != tokenB, "Kswap: IDENTICAL_ADDRESSES");
        (address token0, address token1) =
            tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), "Kswap: ZERO_ADDRESS");
        require(getPair[token0][token1] == address(0), "Kswap: PAIR_EXISTS"); // single check is sufficient
        bytes memory bytecode = type(KswapPair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        KswapPair(pair).initialize(token0, token1);
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse direction
        allPairs.push(pair);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }

    function setFeeTo(address _feeTo) external override {// knownsec 设置 FeeTo 地址、feeToSetter 可用 外部
        require(msg.sender == feeToSetter, "Kswap: FORBIDDEN");
        feeTo = _feeTo;
    }

    function setFeeToSetter(address _feeToSetter) external override {// knownsec 转移 feeToSetter 权限、
feeToSetter 可用 外部
        require(msg.sender == feeToSetter, "Kswap: FORBIDDEN");
        feeToSetter = _feeToSetter;
    }

    function setFeeToRate(uint256 _rate) external override {// knownsec 设置 FeeToRate、feeToSetter 可用 外
部
        require(msg.sender == feeToSetter, "MdexSwapFactory: FORBIDDEN");
        require(_rate > 0, "MdexSwapFactory: FEE_TO_RATE_OVERFLOW");
        feeToRate = _rate.sub(1);
    }
}
```

KswapPair.sol

```
// SPDX-License-Identifier: MIT
```

knownsec

```
pragma solidity =0.6.12;

import "./KswapERC20.sol";
import "../libraries/Math.sol";
import "../libraries/UQ112x112.sol";
import "../interfaces/IERC20.sol";
import "../interfaces/IKswapFactory.sol";
import "../interfaces/IKswapCallee.sol";

contract KswapPair is KswapERC20 {// knownsec  配对合约
    using SafeMathKswap for uint256;
    using UQ112x112 for uint224;

    uint256 public constant MINIMUM_LIQUIDITY = 10**3;
    bytes4 private constant SELECTOR =
        bytes4(keccak256(bytes("transfer(address,uint256)")));

    address public factory;
    address public token0;
    address public token1;

    uint112 private reserve0; // uses single storage slot, accessible via getReserves
    uint112 private reserve1; // uses single storage slot, accessible via getReserves
    uint32 private blockTimestampLast; // uses single storage slot, accessible via getReserves

    uint256 public price0CumulativeLast;
    uint256 public price1CumulativeLast;
    uint256 public kLast; // reserve0 * reserve1, as of immediately after the most recent liquidity event

    uint256 private unlocked = 1;
    modifier lock() {
        require(unlocked == 1, "Kswap: LOCKED");
        unlocked = 0;
        _;
        unlocked = 1;
    }

    function getReserves()
        public
        view
        returns (
            uint112 _reserve0,
            uint112 _reserve1,
            uint32 _blockTimestampLast
        )
    {
        _reserve0 = reserve0;
        _reserve1 = reserve1;
        _blockTimestampLast = blockTimestampLast;
    }

    function _safeTransfer(
        address token,
        address to,
        uint256 value
    ) private {
        (bool success, bytes memory data) =
            token.call(abi.encodeWithSelector(SELECTOR, to, value));
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "Kswap: TRANSFER_FAILED"
        );
    }

    event Mint(address indexed sender, uint256 amount0, uint256 amount1);
    event Burn(
        address indexed sender,
        uint256 amount0,
        uint256 amount1,
        address indexed to
    );
    event Swap(
        address indexed sender,
        uint256 amount0In,
        uint256 amount1In,
        uint256 amount0Out,
        uint256 amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    constructor() public {
        factory = msg.sender;
    }

    // called once by the factory at time of deployment
    function initialize(address _token0, address _token1) external {
```

knownsec

```
        require(msg.sender == factory, "Kswap: FORBIDDEN"); // sufficient check
        token0 = _token0;
        token1 = _token1;
    }

    // update reserves and, on the first call per block, price accumulators
    function _update(
        uint256 balance0,
        uint256 balance1,
        uint112 _reserve0,
        uint112 _reserve1
    ) private {
        require(
            balance0 <= uint112(-1) && balance1 <= uint112(-1),
            "Kswap: OVERFLOW"
        );
        uint32 blockTimestamp = uint32(block.timestamp % 2**32);
        uint32 timeElapsed = blockTimestamp - blockTimestampLast; // overflow is desired
        if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0) {
            // * never overflows, and + overflow is desired
            price0CumulativeLast +=
                uint256(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) *
                timeElapsed;
            price1CumulativeLast +=
                uint256(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) *
                timeElapsed;
        }
        reserve0 = uint112(balance0);
        reserve1 = uint112(balance1);
        blockTimestampLast = blockTimestamp;
        emit Sync(reserve0, reserve1);
    }

    // if fee is on, mint liquidity equivalent to 1/6th of the growth in sqrt(k)
    function _mintFee(uint112 _reserve0, uint112 _reserve1)
        private
        returns (bool feeOn)
    {
        address feeTo = IKswapFactory(factory).feeTo();
        feeOn = feeTo != address(0);
        uint256 _kLast = kLast; // gas savings
        if (feeOn) {
            if (_kLast != 0) {
                uint256 rootK = Math.sqrt(uint256(_reserve0).mul(_reserve1));
                uint256 rootKLast = Math.sqrt(_kLast);
                if (rootK > rootKLast) {
                    uint256 numerator = totalSupply.mul(rootK.sub(rootKLast));
                    uint256 denominator =
                        rootK.mul(IKswapFactory(factory).feeToRate()).add(
                            rootKLast
                        );
                    uint256 liquidity = numerator / denominator;
                    if (liquidity > 0) _mint(feeTo, liquidity);
                }
            }
        } else if (_kLast != 0) {
            kLast = 0;
        }
    }

    // this low-level function should be called from a contract which performs important safety checks
    function mint(address to) external lock returns (uint256 liquidity) {
        (uint112 _reserve0, uint112 _reserve1, ) = getReserves(); // gas savings
        uint256 balance0 = IERC20Kswap(token0).balanceOf(address(this));
        uint256 balance1 = IERC20Kswap(token1).balanceOf(address(this));
        uint256 amount0 = balance0.sub(_reserve0);
        uint256 amount1 = balance1.sub(_reserve1);

        bool feeOn = _mintFee(_reserve0, _reserve1);
        uint256 _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can update in
_mintFee
        if (_totalSupply == 0) {
            liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
            _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first MINIMUM_LIQUIDITY
tokens
        } else {
            liquidity = Math.min(
                amount0.mul(_totalSupply) / _reserve0,
                amount1.mul(_totalSupply) / _reserve1
            );
        }
        require(liquidity > 0, "Kswap: INSUFFICIENT_LIQUIDITY_MINTED");
        _mint(to, liquidity);

        _update(balance0, balance1, _reserve0, _reserve1);
        if (feeOn) kLast = uint256(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
        emit Mint(msg.sender, amount0, amount1);
```

knownsec

```
    }

    // this low-level function should be called from a contract which performs important safety checks
    function burn(address to)
        external
        lock
        returns (uint256 amount0, uint256 amount1)
    {
        (uint112 _reserve0, uint112 _reserve1, ) = getReserves(); // gas savings
        address _token0 = token0; // gas savings
        address _token1 = token1; // gas savings
        uint256 balance0 = IERC20Kswap(_token0).balanceOf(address(this));
        uint256 balance1 = IERC20Kswap(_token1).balanceOf(address(this));
        uint256 liquidity = balanceOf[address(this)];

        bool feeOn = _mintFee(_reserve0, _reserve1);
        uint256 _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can update in
_mintFee
        amount0 = liquidity.mul(balance0) / _totalSupply; // using balances ensures pro-rata distribution
        amount1 = liquidity.mul(balance1) / _totalSupply; // using balances ensures pro-rata distribution
        require(
            amount0 > 0 && amount1 > 0,
            "Kswap: INSUFFICIENT_LIQUIDITY_BURNED"
        );
        _burn(address(this), liquidity);
        _safeTransfer(_token0, to, amount0);
        _safeTransfer(_token1, to, amount1);
        balance0 = IERC20Kswap(_token0).balanceOf(address(this));
        balance1 = IERC20Kswap(_token1).balanceOf(address(this));

        _update(balance0, balance1, _reserve0, _reserve1);
        if (feeOn) kLast = uint256(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
        emit Burn(msg.sender, amount0, amount1, to);
    }

    // this low-level function should be called from a contract which performs important safety checks
    function swap(
        uint256 amount0Out,
        uint256 amount1Out,
        address to,
        bytes calldata data
    ) external lock {
        require(
            amount0Out > 0 || amount1Out > 0,
            "Kswap: INSUFFICIENT_OUTPUT_AMOUNT"
        );
        (uint112 _reserve0, uint112 _reserve1, ) = getReserves(); // gas savings
        require(
            amount0Out < _reserve0 && amount1Out < _reserve1,
            "Kswap: INSUFFICIENT_LIQUIDITY"
        );

        uint256 balance0;
        uint256 balance1;
        {
            // scope for _token{0,1}, avoids stack too deep errors
            address _token0 = token0;
            address _token1 = token1;
            require(to != _token0 && to != _token1, "Kswap: INVALID_TO");
            if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer tokens
            if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer tokens
            if (data.length > 0)
                IKswapCallee(to).KswapCall(
                    msg.sender,
                    amount0Out,
                    amount1Out,
                    data
                );
            balance0 = IERC20Kswap(_token0).balanceOf(address(this));
            balance1 = IERC20Kswap(_token1).balanceOf(address(this));
        }
        uint256 amount0In =
            balance0 > _reserve0 - amount0Out
                ? balance0 - (_reserve0 - amount0Out)
                : 0;
        uint256 amount1In =
            balance1 > _reserve1 - amount1Out
                ? balance1 - (_reserve1 - amount1Out)
                : 0;
        require(
            amount0In > 0 || amount1In > 0,
            "Kswap: INSUFFICIENT_INPUT_AMOUNT"
        );
        {
            // scope for reserve{0,1}Adjusted, avoids stack too deep errors
            uint256 balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
            uint256 balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
```

Knownsec

```
                require(
                    balance0Adjusted.mul(balance1Adjusted) >=
                        uint256(_reserve0).mul(_reserve1).mul(1000**2),
                    "Kswap: K"
                );
            }

            _update(balance0, balance1, _reserve0, _reserve1);
            emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
        }

        // force balances to match reserves
        function skim(address to) external lock {
            address _token0 = token0; // gas savings
            address _token1 = token1; // gas savings
            _safeTransfer(
                _token0,
                to,
                IERC20Kswap(_token0).balanceOf(address(this)).sub(reserve0)
            );
            _safeTransfer(
                _token1,
                to,
                IERC20Kswap(_token1).balanceOf(address(this)).sub(reserve1)
            );
        }

        // force reserves to match balances
        function sync() external lock {
            _update(
                IERC20Kswap(token0).balanceOf(address(this)),
                IERC20Kswap(token1).balanceOf(address(this)),
                reserve0,
                reserve1
            );
        }

        function price(address token, uint256 baseDecimal)
            public
            view
            returns (uint256)
        {
            if (
                (token0 != token && token1 != token) ||
                0 == reserve0 ||
                0 == reserve1
            ) {
                return 0;
            }
            if (token0 == token) {
                return uint256(reserve1).mul(baseDecimal).div(uint256(reserve0));
            } else {
                return uint256(reserve0).mul(baseDecimal).div(uint256(reserve1));
            }
        }
}
```

KswapRouter.sol

```
// SPDX-License-Identifier: MIT

pragma solidity =0.6.12;

import "@openzeppelin/contracts/access/Ownable.sol";
import "../libraries/KswapLibrary.sol";
import "../libraries/SafeMath.sol";
import "../libraries/TransferHelper.sol";
import "../interfaces/IKswapRouter02.sol";
import "../interfaces/IKswapFactory.sol";
import "../interfaces/IERC20.sol";
import "../interfaces/IWOKT.sol";

interface ITradingPool {
    function swap(
        address account,
        address input,
        address output,
        uint256 amount
    ) external returns (bool);
}

contract KswapRouter is IKswapRouter02, Ownable {// knownsec  路由合约
    using SafeMathKswap for uint256;

    address public immutable override factory;
    address public immutable override WOKT;
```

Knownsec

```
    address public override tradingPool;

    modifier ensure(uint256 deadline) {
        require(deadline >= block.timestamp, "KswapRouter: EXPIRED");
        _;
    }

    constructor(address _factory, address _WOKT) public {
        factory = _factory;
        WOKT = _WOKT;
    }

    receive() external payable {
        assert(msg.sender == WOKT); // only accept OKT via fallback from the WOKT contract
    }

    function setTradingPool(address _tradingPool) public onlyOwner {
        tradingPool = _tradingPool;
    }

    // **** ADD LIQUIDITY ****
    function _addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin
    ) internal virtual returns (uint256 amountA, uint256 amountB) {
        // create the pair if it doesn't exist yet
        if (IKswapFactory(factory).getPair(tokenA, tokenB) == address(0)) {
            IKswapFactory(factory).createPair(tokenA, tokenB);
        }
        (uint256 reserveA, uint256 reserveB) =
            KswapLibrary.getReserves(factory, tokenA, tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint256 amountBOptimal =
                KswapLibrary.quote(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(
                    amountBOptimal >= amountBMin,
                    "KswapRouter: INSUFFICIENT_B_AMOUNT"
                );
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint256 amountAOptimal =
                    KswapLibrary.quote(amountBDesired, reserveB, reserveA);
                assert(amountAOptimal <= amountADesired);
                require(
                    amountAOptimal >= amountAMin,
                    "KswapRouter: INSUFFICIENT_A_AMOUNT"
                );
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    )
        external
        virtual
        override
        ensure(deadline)
        returns (
            uint256 amountA,
            uint256 amountB,
            uint256 liquidity
        )
    {
        (amountA, amountB) = _addLiquidity(
            tokenA,
            tokenB,
            amountADesired,
            amountBDesired,
            amountAMin,
            amountBMin
        );
```

Knownsec

```
        address pair = KswapLibrary.pairFor(factory, tokenA, tokenB);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = IKswapPair(pair).mint(to);
    }

    function addLiquidityETH(
        address token,
        uint256 amountTokenDesired,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    )
        external
        payable
        virtual
        override
        ensure(deadline)
        returns (
            uint256 amountToken,
            uint256 amountETH,
            uint256 liquidity
        )
    {
        (amountToken, amountETH) = _addLiquidity(
            token,
            WOKT,
            amountTokenDesired,
            msg.value,
            amountTokenMin,
            amountETHMin
        );
        address pair = KswapLibrary.pairFor(factory, token, WOKT);
        TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
        IWOKT(WOKT).deposit{value: amountETH}();
        assert(IWOKT(WOKT).transfer(pair, amountETH));
        liquidity = IKswapPair(pair).mint(to);
        // refund dust eth, if any
        if (msg.value > amountETH)
            TransferHelper.safeTransferETH(msg.sender, msg.value - amountETH);
    }

    // **** REMOVE LIQUIDITY ****
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    )
        public
        virtual
        override
        ensure(deadline)
        returns (uint256 amountA, uint256 amountB)
    {
        address pair = KswapLibrary.pairFor(factory, tokenA, tokenB);
        IKswapPair(pair).transferFrom(msg.sender, pair, liquidity); // send liquidity to pair
        (uint256 amount0, uint256 amount1) = IKswapPair(pair).burn(to);
        (address token0, ) = KswapLibrary.sortTokens(tokenA, tokenB);
        (amountA, amountB) = tokenA == token0
            ? (amount0, amount1)
            : (amount1, amount0);
        require(amountA >= amountAMin, "KswapRouter: INSUFFICIENT_A_AMOUNT");
        require(amountB >= amountBMin, "KswapRouter: INSUFFICIENT_B_AMOUNT");
    }

    function removeLiquidityETH(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    )
        public
        virtual
        override
        ensure(deadline)
        returns (uint256 amountToken, uint256 amountETH)
    {
        (amountToken, amountETH) = removeLiquidity(
            token,
            WOKT,
```

Knownsec

```
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, amountToken);
        IWOKT(WOKT).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }

    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external virtual override returns (uint256 amountA, uint256 amountB) {
        address pair = KswapLibrary.pairFor(factory, tokenA, tokenB);
        uint256 value = approveMax ? uint256(-1) : liquidity;
        IKswapPair(pair).permit(
            msg.sender,
            address(this),
            value,
            deadline,
            v,
            r,
            s
        );
        (amountA, amountB) = removeLiquidity(
            tokenA,
            tokenB,
            liquidity,
            amountAMin,
            amountBMin,
            to,
            deadline
        );
    }

    function removeLiquidityETHWithPermit(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    )
        external
        virtual
        override
        returns (uint256 amountToken, uint256 amountETH)
    {
        address pair = KswapLibrary.pairFor(factory, token, WOKT);
        uint256 value = approveMax ? uint256(-1) : liquidity;
        IKswapPair(pair).permit(
            msg.sender,
            address(this),
            value,
            deadline,
            v,
            r,
            s
        );
        (amountToken, amountETH) = removeLiquidityETH(
            token,
            liquidity,
            amountTokenMin,
            amountETHMin,
            to,
            deadline
        );
    }

    // **** REMOVE LIQUIDITY (supporting fee-on-transfer tokens) ****
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
```

Knownsec

```
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    ) public virtual override ensure(deadline) returns (uint256 amountETH) {
        (, amountETH) = removeLiquidity(
            token,
            WOKT,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(
            token,
            to,
            IERC20Kswap(token).balanceOf(address(this))
        );
        IWOKT(WOKT).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }

    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external virtual override returns (uint256 amountETH) {
        address pair = KswapLibrary.pairFor(factory, token, WOKT);
        uint256 value = approveMax ? uint256(-1) : liquidity;
        IKswapPair(pair).permit(
            msg.sender,
            address(this),
            value,
            deadline,
            v,
            r,
            s
        );
        amountETH = removeLiquidityETHSupportingFeeOnTransferTokens(
            token,
            liquidity,
            amountTokenMin,
            amountETHMin,
            to,
            deadline
        );
    }

    // **** SWAP ****
    // requires the initial amount to have already been sent to the first pair
    function _swap(
        uint256[] memory amounts,
        address[] memory path,
        address _to
    ) internal virtual {
        for (uint256 i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0, ) = KswapLibrary.sortTokens(input, output);
            uint256 amountOut = amounts[i + 1];
            if (tradingPool != address(0)) {
                ITradingPool(tradingPool).swap(
                    msg.sender,
                    input,
                    output,
                    amountOut
                );
            }
            (uint256 amount0Out, uint256 amount1Out) =
                input == token0
                    ? (uint256(0), amountOut)
                    : (amountOut, uint256(0));
            address to =
                i < path.length - 2
                    ? KswapLibrary.pairFor(factory, output, path[i + 2])
                    : _to;
            IKswapPair(KswapLibrary.pairFor(factory, input, output)).swap(
                amount0Out,
                amount1Out,
```

Knownsec

```
                to,
                new bytes(0)
            );
        }
    }

    function swapExactTokensForTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    )
        external
        virtual
        override
        ensure(deadline)
        returns (uint256[] memory amounts)
    {
        amounts = KswapLibrary.getAmountsOut(factory, amountIn, path);
        require(
            amounts[amounts.length - 1] >= amountOutMin,
            "KswapRouter: INSUFFICIENT_OUTPUT_AMOUNT"
        );
        TransferHelper.safeTransferFrom(
            path[0],
            msg.sender,
            KswapLibrary.pairFor(factory, path[0], path[1]),
            amounts[0]
        );
        _swap(amounts, path, to);
    }

    function swapTokensForExactTokens(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
        uint256 deadline
    )
        external
        virtual
        override
        ensure(deadline)
        returns (uint256[] memory amounts)
    {
        amounts = KswapLibrary.getAmountsIn(factory, amountOut, path);
        require(
            amounts[0] <= amountInMax,
            "KswapRouter: EXCESSIVE_INPUT_AMOUNT"
        );
        TransferHelper.safeTransferFrom(
            path[0],
            msg.sender,
            KswapLibrary.pairFor(factory, path[0], path[1]),
            amounts[0]
        );
        _swap(amounts, path, to);
    }

    function swapExactETHForTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    )
        external
        payable
        virtual
        override
        ensure(deadline)
        returns (uint256[] memory amounts)
    {
        require(path[0] == WOKT, "KswapRouter: INVALID_PATH");
        amounts = KswapLibrary.getAmountsOut(factory, msg.value, path);
        require(
            amounts[amounts.length - 1] >= amountOutMin,
            "KswapRouter: INSUFFICIENT_OUTPUT_AMOUNT"
        );
        IWOKT(WOKT).deposit{value: amounts[0]}();
        assert(
            IWOKT(WOKT).transfer(
                KswapLibrary.pairFor(factory, path[0], path[1]),
                amounts[0]
            )
        );
        _swap(amounts, path, to);
```

Knownsec

```
    }

    function swapTokensForExactETH(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
        uint256 deadline
    )
        external
        virtual
        override
        ensure(deadline)
        returns (uint256[] memory amounts)
    {
        require(path[path.length - 1] == WOKT, "KswapRouter: INVALID_PATH");
        amounts = KswapLibrary.getAmountsIn(factory, amountOut, path);
        require(
            amounts[0] <= amountInMax,
            "KswapRouter: EXCESSIVE_INPUT_AMOUNT"
        );
        TransferHelper.safeTransferFrom(
            path[0],
            msg.sender,
            KswapLibrary.pairFor(factory, path[0], path[1]),
            amounts[0]
        );
        _swap(amounts, path, address(this));
        IWOKT(WOKT).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }

    function swapExactTokensForETH(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    )
        external
        virtual
        override
        ensure(deadline)
        returns (uint256[] memory amounts)
    {
        require(path[path.length - 1] == WOKT, "KswapRouter: INVALID_PATH");
        amounts = KswapLibrary.getAmountsOut(factory, amountIn, path);
        require(
            amounts[amounts.length - 1] >= amountOutMin,
            "KswapRouter: INSUFFICIENT_OUTPUT_AMOUNT"
        );
        TransferHelper.safeTransferFrom(
            path[0],
            msg.sender,
            KswapLibrary.pairFor(factory, path[0], path[1]),
            amounts[0]
        );
        _swap(amounts, path, address(this));
        IWOKT(WOKT).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }

    function swapETHForExactTokens(
        uint256 amountOut,
        address[] calldata path,
        address to,
        uint256 deadline
    )
        external
        payable
        virtual
        override
        ensure(deadline)
        returns (uint256[] memory amounts)
    {
        require(path[0] == WOKT, "KswapRouter: INVALID_PATH");
        amounts = KswapLibrary.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= msg.value, "KswapRouter: EXCESSIVE_INPUT_AMOUNT");
        IWOKT(WOKT).deposit{value: amounts[0]}();
        assert(
            IWOKT(WOKT).transfer(
                KswapLibrary.pairFor(factory, path[0], path[1]),
                amounts[0]
            )
        );
        _swap(amounts, path, to);
        // refund dust eth, if any
```

Knownsec

```
        if (msg.value > amounts[0])
            TransferHelper.safeTransferETH(msg.sender, msg.value - amounts[0]);
    }

    // **** SWAP (supporting fee-on-transfer tokens) ****
    // requires the initial amount to have already been sent to the first pair
    function _swapSupportingFeeOnTransferTokens(
        address[] memory path,
        address _to
    ) internal virtual {
        for (uint256 i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0, ) = KswapLibrary.sortTokens(input, output);
            IKswapPair pair =
                IKswapPair(KswapLibrary.pairFor(factory, input, output));
            uint256 amountInput;
            uint256 amountOutput;
            {
                // scope to avoid stack too deep errors
                (uint256 reserve0, uint256 reserve1, ) = pair.getReserves();
                (uint256 reserveInput, uint256 reserveOutput) =
                    input == token0
                        ? (reserve0, reserve1)
                        : (reserve1, reserve0);
                amountInput = IERC20Kswap(input).balanceOf(address(pair)).sub(
                    reserveInput
                );
                amountOutput = KswapLibrary.getAmountOut(
                    amountInput,
                    reserveInput,
                    reserveOutput
                );
            }
            if (tradingPool != address(0)) {
                ITradingPool(tradingPool).swap(
                    msg.sender,
                    input,
                    output,
                    amountOutput
                );
            }
            (uint256 amount0Out, uint256 amount1Out) =
                input == token0
                    ? (uint256(0), amountOutput)
                    : (amountOutput, uint256(0));
            address to =
                i < path.length - 2
                    ? KswapLibrary.pairFor(factory, output, path[i + 2])
                    : _to;
            pair.swap(amount0Out, amount1Out, to, new bytes(0));
        }
    }

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external virtual override ensure(deadline) {
        TransferHelper.safeTransferFrom(
            path[0],
            msg.sender,
            KswapLibrary.pairFor(factory, path[0], path[1]),
            amountIn
        );
        uint256 balanceBefore =
            IERC20Kswap(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20Kswap(path[path.length - 1]).balanceOf(to).sub(
                balanceBefore
            ) >= amountOutMin,
            "KswapRouter: INSUFFICIENT_OUTPUT_AMOUNT"
        );
    }

    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable virtual override ensure(deadline) {
        require(path[0] == WOKT, "KswapRouter: INVALID_PATH");
        uint256 amountIn = msg.value;
        IWOKT(WOKT).deposit{value: amountIn}();
        assert(
```

Knownsec

```
                IWOKT(WOKT).transfer(
                    KswapLibrary.pairFor(factory, path[0], path[1]),
                    amountIn
                )
            );
            uint256 balanceBefore =
                IERC20Kswap(path[path.length - 1]).balanceOf(to);
            _swapSupportingFeeOnTransferTokens(path, to);
            require(
                IERC20Kswap(path[path.length - 1]).balanceOf(to).sub(
                    balanceBefore
                ) >= amountOutMin,
                "KswapRouter: INSUFFICIENT_OUTPUT_AMOUNT"
            );
        }

        function swapExactTokensForETHSupportingFeeOnTransferTokens(
            uint256 amountIn,
            uint256 amountOutMin,
            address[] calldata path,
            address to,
            uint256 deadline
        ) external virtual override ensure(deadline) {
            require(path[path.length - 1] == WOKT, "KswapRouter: INVALID_PATH");
            TransferHelper.safeTransferFrom(
                path[0],
                msg.sender,
                KswapLibrary.pairFor(factory, path[0], path[1]),
                amountIn
            );
            _swapSupportingFeeOnTransferTokens(path, address(this));
            uint256 amountOut = IERC20Kswap(WOKT).balanceOf(address(this));
            require(
                amountOut >= amountOutMin,
                "KswapRouter: INSUFFICIENT_OUTPUT_AMOUNT"
            );
            IWOKT(WOKT).withdraw(amountOut);
            TransferHelper.safeTransferETH(to, amountOut);
        }

        // **** LIBRARY FUNCTIONS ****
        function quote(
            uint256 amountA,
            uint256 reserveA,
            uint256 reserveB
        ) public pure virtual override returns (uint256 amountB) {
            return KswapLibrary.quote(amountA, reserveA, reserveB);
        }

        function getAmountOut(
            uint256 amountIn,
            uint256 reserveIn,
            uint256 reserveOut
        ) public pure virtual override returns (uint256 amountOut) {
            return KswapLibrary.getAmountOut(amountIn, reserveIn, reserveOut);
        }

        function getAmountIn(
            uint256 amountOut,
            uint256 reserveIn,
            uint256 reserveOut
        ) public pure virtual override returns (uint256 amountIn) {
            return KswapLibrary.getAmountIn(amountOut, reserveIn, reserveOut);
        }

        function getAmountsOut(uint256 amountIn, address[] memory path)
            public
            view
            virtual
            override
            returns (uint256[] memory amounts)
        {
            return KswapLibrary.getAmountsOut(factory, amountIn, path);
        }

        function getAmountsIn(uint256 amountOut, address[] memory path)
            public
            view
            virtual
            override
            returns (uint256[] memory amounts)
        {
            return KswapLibrary.getAmountsIn(factory, amountOut, path);
        }
    }
```

Knownsec

TimeLock.sol

```
// SPDX-License-Identifier: MIT

pragma solidity =0.6.12;

import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";

contract TeamTimeLock {// knownsec 治理 提案生效时间锁
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    IERC20 public token;
    uint256 public constant PERIOD = 30 days;
    uint256 public constant CYCLE_TIMES = 24;
    uint256 public fixedQuantity; // Monthly rewards are fixed
    uint256 public startTime;
    uint256 public delay;
    uint256 public cycle; // cycle already received
    uint256 public hasReward; // Rewards already withdrawn
    address public beneficiary;
    string public introduce;

    event WithDraw(
        address indexed operator,
        address indexed to,
        uint256 amount
    );

    constructor(
        address _beneficiary,
        address _token,
        uint256 _fixedQuantity,
        uint256 _startTime,
        uint256 _delay,
        string memory _introduce
    ) public {
        require(
            _beneficiary != address(0) && _token != address(0),
            "TimeLock: zero address"
        );
        require(_fixedQuantity > 0, "TimeLock: fixedQuantity is zero");
        beneficiary = _beneficiary;
        token = IERC20(_token);
        fixedQuantity = _fixedQuantity;
        delay = _delay;
        startTime = _startTime.add(_delay);
        introduce = _introduce;
    }

    function getBalance() public view returns (uint256) {
        return token.balanceOf(address(this));
    }

    function getReward() public view returns (uint256) {
        // Has ended or not started
        if (cycle >= CYCLE_TIMES || block.timestamp <= startTime) {
            return 0;
        }
        uint256 pCycle = (block.timestamp.sub(startTime)).div(PERIOD);
        if (pCycle >= CYCLE_TIMES) {
            return token.balanceOf(address(this));
        }
        return pCycle.sub(cycle).mul(fixedQuantity);
    }

    function withDraw() external {
        uint256 reward = getReward();
        require(reward > 0, "TimeLock: no reward");
        uint256 pCycle = (block.timestamp.sub(startTime)).div(PERIOD);
        cycle = pCycle >= CYCLE_TIMES ? CYCLE_TIMES : pCycle;
        hasReward = hasReward.add(reward);
        token.safeTransfer(beneficiary, reward);
        emit WithDraw(msg.sender, beneficiary, reward);
    }

    // Update beneficiary address by the previous beneficiary.
    function setBeneficiary(address _newBeneficiary) public {
        require(msg.sender == beneficiary, "Not beneficiary");
        beneficiary = _newBeneficiary;
    }
}
```

knownsec

6. Appendix B: Vulnerability rating standard

Smart contract vulnerability rating standards	
Level	Level Description
High	Vulnerabilities that can directly cause the loss of token contracts or user funds, such as: value overflow loopholes that can cause the value of tokens to zero, fake recharge loopholes that can cause exchanges to lose tokens, and can cause contract accounts to lose OKT or tokens. Access loopholes, etc.; Vulnerabilities that can cause loss of ownership of token contracts, such as: access control defects of key functions, call injection leading to bypassing of access control of key functions, etc.; Vulnerabilities that can cause the token contract to not work properly, such as: denial of service vulnerability caused by sending OKT to malicious addresses, and denial of service vulnerability caused by exhaustion of gas.
Medium	High-risk vulnerabilities that require specific addresses to trigger, such as value overflow vulnerabilities that can be triggered by token contract owners; access control defects for non-critical functions, and logical design defects that cannot cause direct capital losses, etc.
Low	Vulnerabilities that are difficult to be triggered, vulnerabilities with limited damage after triggering, such as value overflow vulnerabilities that require a large amount of

Knownsec

	OKT or tokens to trigger, vulnerabilities where attackers cannot directly profit after triggering value overflow, and the transaction sequence triggered by specifying high gas depends on the risk Wait.

Knownsec

7. Appendix C：Introduction to auditing tools

7.1 Manticore

Manticore is a symbolic execution tool for analyzing binary files and smart contracts. Manticore includes a symbolic Ethereum Virtual Machine (EVM), an EVM disassembler/assembler and a convenient interface for automatic compilation and analysis of Solidity. It also integrates Ethersplay, Bit of Traits of Bits visual disassembler for EVM bytecode, used for visual analysis. Like binary files, Manticore provides a simple command line interface and a Python for analyzing EVM bytecode API.

7.2 Oyente

Oyente is a smart contract analysis tool. Oyente can be used to detect common bugs in smart contracts, such as reentrancy, transaction sequencing dependencies, etc. More convenient, Oyente's design is modular, so this allows advanced users to implement and Insert their own detection logic to check the custom attributes in their contract.

7.3 securify.sh

Securify can verify common security issues of Ethereum smart contracts, such as disordered transactions and lack of input verification. It analyzes all possible execution paths of the program while fully automated. In addition, Securify also has a

Knownsec

specific language for specifying vulnerabilities, which makes Securify can keep an eye on current security and other reliability issues at any time.

7.4 Echidna

Echidna is a Haskell library designed for fuzzing EVM code.

7.5 MAIAN

MAIAN is an automated tool for finding vulnerabilities in Ethereum smart contracts. Maian processes the bytecode of the contract and tries to establish a series of transactions to find and confirm the error.

7.6 ethersplay

ethersplay is an EVM disassembler, which contains relevant analysis tools.

7.7 ida-evm

ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

7.8 Remix-ide

ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

knownsec

7.9 Knownsec Penetration Tester Special Toolkit

Pen-Tester tools collection is created by KnownSec team. It contains plenty of Pen-Testing tools such as automatic testing tool, scripting tool, Self-developed tools etc.

Knownsec

KNOWNSEC

Beijing KnownSec Information Technology Co., Ltd.

Advisory telephone	+86(10)400 060 9587
E-mail	sec@knownsec.com
Website	www.knownsec.com
Address	wangjing soho T2-B2509,Chaoyang District, Beijing