Smart Contract Security Audit Report

The SlowMist Security Team received the SIX team's application for smart contract security audit of the SIX on Jun 17, 2020. The following are the details and results of this smart contract security audit:

Token name :

SIX

File name and HASH(SHA256) :

8_SIX.sol.zip: 7541082bb461d0a0039eaaf9377ab50ac571bace3edbe87fc8c20ff93ad75f62

7_contract.zip: 5226305ddfbec7a859230a51ef247df114f9a380f579da58f50b8d3ebfbb8643

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audit	-	Passed
3	Authority Control Audit	Permission vulnerability audit	Passed
		Excessive auditing authority	Passed
4	Safety Design Audit	Zeppelin module safe use	Passed
		Compiler version security	Passed
		Hard-coded address security	Passed
		Fallback function safe use	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed
6	Gas Optimization Audit	-	Passed
7	Design Logic Audit	-	Passed
8	“False Deposit” vulnerability Audit	-	Passed
9	Malicious Event Log Audit	-	Passed

10	Scoping and Declarations Audit	-	Passed
11	Replay Attack Audit	ECDSA's Signature Replay Audit	Passed
12	Uninitialized Storage Pointers Audit	-	Passed
13	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : **Passed**

Audit Number : 0X002006220001

Audit Date : Jun 22, 2020

Audit Team : SlowMist Security Team

(Statement : SlowMist only issues this report based on the fact that has occurred or existed before the report is issued, and bears the corresponding responsibility in this regard. For the facts occur or exist later after the report, SlowMist cannot judge the security status of its smart contract. SlowMist is not responsible for it. The security audit analysis and other contents of this report are based on the documents and materials provided by the information provider to SlowMist as of the date of this report (referred to as "the provided information"). SlowMist assumes that: there has been no information missing, tampered, deleted, or concealed. If the information provided has been missed, modified, deleted, concealed or reflected and is inconsistent with the actual situation, SlowMist will not bear any responsibility for the resulting loss and adverse effects. SlowMist will not bear any responsibility for the background or other circumstances of the project.)

Summary: This is a token contract that does not contain the tokenVault section. The total amount of contract tokens can be changed, users can burn their tokens through the burn function. Minter can mint unlimited tokens through the mint function. OpenZeppelin’s SafeMath security module is used, which is a commendable approach. The contract does not have the Overflow and the Race Conditions issue.

The source code:

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue

pragma solidity ^0.5.0;

library Roles {
    struct Role {
        mapping (address => bool) bearer;
    }

    /**
```

```
     * @dev Give an account access to this role.
     */
    function add(Role storage role, address account) internal {
        require(!has(role, account), "Roles: account already has role");
        role.bearer[account] = true;
    }

    /**
     * @dev Remove an account's access to this role.
     */
    function remove(Role storage role, address account) internal {
        require(has(role, account), "Roles: account does not have role");
        role.bearer[account] = false;
    }

    /**
     * @dev Check if an account has this role.
     * @return bool
     */
    function has(Role storage role, address account) internal view returns (bool) {
        require(account != address(0), "Roles: account is the zero address");
        return role.bearer[account];
    }
}

contract MinterRole {
    using Roles for Roles.Role;

    event MinterAdded(address indexed account);
    event MinterRemoved(address indexed account);

    Roles.Role private _minters;

    constructor () internal {
        _addMinter(msg.sender);
    }

    modifier onlyMinter() {
        require(isMinter(msg.sender), "MinterRole: caller does not have the Minter role");
        _;
    }
```

```
    function isMinter(address account) public view returns (bool) {
        return _minters.has(account);
    }

    function addMinter(address account) public onlyMinter {
        _addMinter(account);
    }

    function renounceMinter() public {
        _removeMinter(msg.sender);
    }

    function _addMinter(address account) internal {
        _minters.add(account);
        emit MinterAdded(account);
    }

    function _removeMinter(address account) internal {
        _minters.remove(account);
        emit MinterRemoved(account);
    }
}

contract PauserRole {
    using Roles for Roles.Role;

    event PauserAdded(address indexed account);
    event PauserRemoved(address indexed account);

    Roles.Role private _pausers;

    constructor () internal {
        _addPauser(msg.sender);
    }

    modifier onlyPauser() {
        require(isPauser(msg.sender), "PauserRole: caller does not have the Pauser role");
        _;
    }

    function isPauser(address account) public view returns (bool) {
        return _pausers.has(account);
```

```
    }

    function addPauser(address account) public onlyPauser {
        _addPauser(account);
    }

    function renouncePauser() public {
        _removePauser(msg.sender);
    }

    function _addPauser(address account) internal {
        _pausers.add(account);
        emit PauserAdded(account);
    }

    function _removePauser(address account) internal {
        _pausers.remove(account);
        emit PauserRemoved(account);
    }
}

interface IKIP13 {
    /**
     * @dev Returns true if this contract implements the interface defined by
     * `interfaceId`. See the corresponding
     * [KIP-13
section](http://kips.klaytn.com/KIPs/kip-13-interface_query_standard#how-interface-identifiers-are-defined)
     * to learn more about how these ids are created.
     *
     * This function call must use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool);
}

contract KIP13 is IKIP13 {
    /*
     * bytes4(keccak256('supportsInterface(bytes4)')) == 0x01ffc9a7
     */
    bytes4 private constant _INTERFACE_ID_KIP13 = 0x01ffc9a7;

    /**
     * @dev Mapping of interface ids to whether or not it's supported.
```

```
     */
    mapping(bytes4 => bool) private _supportedInterfaces;

    constructor () internal {
        // Derived contracts need only register support for their own interfaces,
        // we register support for KIP13 itself here
        _registerInterface(_INTERFACE_ID_KIP13);
    }

    /**
     * @dev See `IKIP13.supportsInterface`.
     *
     * Time complexity O(1), guaranteed to always use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool) {
        return _supportedInterfaces[interfaceId];
    }

    /**
     * @dev Registers the contract as an implementer of the interface defined by
     * `interfaceId`. Support of the actual KIP13 interface is automatic and
     * registering its interface id is not required.
     *
     * See `IKIP13.supportsInterface`.
     *
     * Requirements:
     *
     * - `interfaceId` cannot be the KIP13 invalid interface (`0xffffffff`).
     */
    function _registerInterface(bytes4 interfaceId) internal {
        require(interfaceId != 0xffffffff, "KIP13: invalid interface id");
        _supportedInterfaces[interfaceId] = true;
    }
}

contract Pausable is PauserRole {
    /**
     * @dev Emitted when the pause is triggered by a pauser (`account`).
     */
    event Paused(address account);

    /**
```

```
 * @dev Emitted when the pause is lifted by a pauser (`account`).
 */
event Unpaused(address account);

bool private _paused;

/**
 * @dev Initializes the contract in unpaused state. Assigns the Pauser role
 * to the deployer.
 */
constructor () internal {
    _paused = false;
}

/**
 * @dev Returns true if the contract is paused, and false otherwise.
 */
function paused() public view returns (bool) {
    return _paused;
}

/**
 * @dev Modifier to make a function callable only when the contract is not paused.
 */
modifier whenNotPaused() {
    require(!_paused, "Pausable: paused");
    _;
}

/**
 * @dev Modifier to make a function callable only when the contract is paused.
 */
modifier whenPaused() {
    require(_paused, "Pausable: not paused");
    _;
}

/**
 * @dev Called by a pauser to pause, triggers stopped state.
 */
```

//SlowMist// Suspending all transactions upon major abnormalities is a recommended approach

```
    function pause() public onlyPauser whenNotPaused {
        _paused = true;
        emit Paused(msg.sender);
    }

    /**
     * @dev Called by a pauser to unpause, returns to normal state.
     */
    function unpause() public onlyPauser whenPaused {
        _paused = false;
        emit Unpaused(msg.sender);
    }
}

//SlowMist// OpenZeppelin's SafeMath security Module is used, which is a recommend approach

library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
```

```
        require(b <= a, "SafeMath: subtraction overflow");
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
```

```
        require(b > 0, "SafeMath: division by zero");
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0, "SafeMath: modulo by zero");
        return a % b;
    }
}

contract IKIP7 is IKIP13 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a `Transfer` event.
     */
```

```
function transfer(address recipient, uint256 amount) external returns (bool);

/**
 * @dev Returns the remaining number of tokens that `spender` will be
 * allowed to spend on behalf of `owner` through `transferFrom`. This is
 * zero by default.
 *
 * This value changes when `approve` or `transferFrom` are called.
 */
function allowance(address owner, address spender) external view returns (uint256);

/**
 * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * > Beware that changing an allowance with this method brings the risk
 * that someone may use both the old and the new allowance by unfortunate
 * transaction ordering. One possible solution to mitigate this race
 * condition is to first reduce the spender's allowance to 0 and set the
 * desired value afterwards:
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
 *
 * Emits an `Approval` event.
 */
function approve(address spender, uint256 amount) external returns (bool);

/**
 * @dev Moves `amount` tokens from `sender` to `recipient` using the
 * allowance mechanism. `amount` is then deducted from the caller's
 * allowance.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * Emits a `Transfer` event.
 */
function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

/**
* @dev Moves `amount` tokens from the caller's account to `recipient`.
*/
function safeTransfer(address recipient, uint256 amount, bytes memory data) public;
```

```
    /**
     * @dev   Moves `amount` tokens from the caller's account to `recipient`.
     */
    function safeTransfer(address recipient, uint256 amount) public;

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the allowance mechanism.
     * `amount` is then deducted from the caller's allowance.
     */
    function safeTransferFrom(address sender, address recipient, uint256 amount, bytes memory data) public;

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the allowance mechanism.
     * `amount` is then deducted from the caller's allowance.
     */
    function safeTransferFrom(address sender, address recipient, uint256 amount) public;

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to `approve`. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

contract IKIP7Receiver {
    /**
     * @notice Handle the receipt of KIP-7 token
     * @dev The KIP-7 smart contract calls this function on the recipient
     *   after a `safeTransfer`. This function MAY throw to revert and reject the
     *   transfer. Return of other than the magic value MUST result in the
     *   transaction being reverted.
     *   Note: the contract address is always the message sender.
     * @param _operator The address which called `safeTransferFrom` function
```

```
     * @param _from The address which previously owned the token
     * @param _amount The token amount which is being transferred.
     * @param _data Additional data with no specified format
     * @return `bytes4(keccak256("onKIP7Received(address,address,uint256,bytes)"))`
     *   unless throwing
     */
    function onKIP7Received(address _operator, address _from, uint256 _amount, bytes memory _data) public returns
(bytes4);
}

contract KIP7 is KIP13, IKIP7 {
    using SafeMath for uint256;
    using Address for address;

    // Equals to `bytes4(keccak256("onKIP7Received(address,address,uint256,bytes)"))`
    // which can be also obtained as `IKIP7Receiver(0).onKIP7Received.selector`
    bytes4 private constant _KIP7_RECEIVED = 0x9d188c22;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowances;

    uint256 private _totalSupply;

    /*
     *     bytes4(keccak256('totalSupply()')) == 0x18160ddd
     *     bytes4(keccak256('balanceOf(address)')) == 0x70a08231
     *     bytes4(keccak256('transfer(address,uint256)')) == 0xa9059cbb
     *     bytes4(keccak256('allowance(address,address)')) == 0xdd62ed3e
     *     bytes4(keccak256('approve(address,uint256)')) == 0x095ea7b3
     *     bytes4(keccak256('transferFrom(address,address,uint256)')) == 0x23b872dd
     *     bytes4(keccak256("safeTransfer(address,uint256)")) == 0x423f6cef
     *     bytes4(keccak256("safeTransfer(address,uint256,bytes)")) == 0xeb795549
     *     bytes4(keccak256("safeTransferFrom(address,address,uint256)")) == 0x42842e0e
     *     bytes4(keccak256("safeTransferFrom(address,address,uint256,bytes)")) == 0xb88d4fde
     *
     *     => 0x18160ddd ^ 0x70a08231 ^ 0xa9059cbb ^ 0xdd62ed3e ^ 0x095ea7b3 ^ 0x23b872dd ^ 0x423f6cef ^
0xeb795549 ^ 0x42842e0e ^ 0xb88d4fde == 0x65787371
     */
    bytes4 private constant _INTERFACE_ID_KIP7 = 0x65787371;

    constructor () public {
```

```
        // register the supported interfaces to conform to KIP7 via KIP13
        _registerInterface(_INTERFACE_ID_KIP7);
    }

    /**
     * @dev See `IKIP7.totalSupply`.
     */
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev See `IKIP7.balanceOf`.
     */
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    /**
     * @dev See `IKIP7.transfer`.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address recipient, uint256 amount) public returns (bool) {
        _transfer(msg.sender, recipient, amount);

        return true; //SlowMist// The return value conforms to the KIP7 specification

    }

    /**
     * @dev See `IKIP7.allowance`.
     */
    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowances[owner][spender];
    }

    /**
     * @dev See `IKIP7.approve`.
     *
```

```
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 value) public returns (bool) {
        _approve(msg.sender, spender, value);

        return true; //SlowMist// The return value conforms to the KIP7 specification

    }

    /**
     * @dev See `IKIP7.transferFrom`.
     *
     * Emits an `Approval` event indicating the updated allowance. This is not
     * required by the KIP. See the note at the beginning of `KIP7`;
     *
     * Requirements:
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `value`.
     * - the caller must have allowance for `sender`'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, msg.sender, _allowances[sender][msg.sender].sub(amount));

        return true; //SlowMist// The return value conforms to the KIP7 specification

    }

    /**
    * @dev   Moves `amount` tokens from the caller's account to `recipient`.
    */
    function safeTransfer(address recipient, uint256 amount) public {
        safeTransfer(recipient, amount, "");
    }

    /**
    * @dev Moves `amount` tokens from the caller's account to `recipient`.
    */
    function safeTransfer(address recipient, uint256 amount, bytes memory data) public {
        transfer(recipient, amount);
```

```
        require(_checkOnKIP7Received(msg.sender, recipient, amount, data), "KIP7: transfer to non KIP7Receiver implementer");
    }

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the allowance mechanism.
     * `amount` is then deducted from the caller's allowance.
     */
    function safeTransferFrom(address sender, address recipient, uint256 amount) public {
        safeTransferFrom(sender, recipient, amount, "");
    }

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the allowance mechanism.
     * `amount` is then deducted from the caller's allowance.
     */
    function safeTransferFrom(address sender, address recipient, uint256 amount, bytes memory data) public {
        transferFrom(sender, recipient, amount);
        require(_checkOnKIP7Received(sender, recipient, amount, data), "KIP7: transfer to non KIP7Receiver implementer");
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to `transfer`, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a `Transfer` event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "KIP7: transfer from the zero address");

        require(recipient != address(0), "KIP7: transfer to the zero address"); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer
```

```
        _balances[sender] = _balances[sender].sub(amount);
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a `Transfer` event with `from` set to the zero address.
     *
     * Requirements
     *
     * - `to` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "KIP7: mint to the zero address");

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }

     /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a `Transfer` event with `to` set to the zero address.
     *
     * Requirements
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 value) internal {
        require(account != address(0), "KIP7: burn from the zero address");

        _totalSupply = _totalSupply.sub(value);
        _balances[account] = _balances[account].sub(value);
        emit Transfer(account, address(0), value);
    }

    /**
```

```
    * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
     *
     * This internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an `Approval` event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(address owner, address spender, uint256 value) internal {
        require(owner != address(0), "KIP7: approve from the zero address");

        require(spender != address(0), "KIP7: approve to the zero address"); //SlowMist// This kind of check is very good, avoiding user mistake leading to approve errors

        _allowances[owner][spender] = value;
        emit Approval(owner, spender, value);
    }

    /**
     * @dev Destroys `amount` tokens from `account`.`amount` is then deducted
     * from the caller's allowance.
     *
     * See `_burn` and `_approve`.
     */
    //SlowMist// Because _burnFrom() and transferFrom() share the _allowances amount of _approve(), if the agent be evil, there is the possibility of malicious burn
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount));
    }

    /**
     * @dev Internal function to invoke `onKIP7Received` on a target address.
     * The call is not executed if the target address is not a contract.
     */
```

```
    function _checkOnKIP7Received(address sender, address recipient, uint256 amount, bytes memory _data)
        internal returns (bool)
    {
        if (!recipient.isContract()) {
            return true;
        }

        bytes4 retval = IKIP7Receiver(recipient).onKIP7Received(msg.sender, sender, amount, _data);
        return (retval == _KIP7_RECEIVED);
    }
}

contract KIP7Burnable is KIP13, KIP7 {
    /*
     *     bytes4(keccak256('burn(uint256)')) == 0x42966c68
     *     bytes4(keccak256('burnFrom(address,uint256)')) == 0x79cc6790
     *
     *     => 0x42966c68 ^ 0x79cc6790 == 0x3b5a0bf8
     */
    bytes4 private constant _INTERFACE_ID_KIP7BURNABLE = 0x3b5a0bf8;

    constructor () public {
        // register the supported interfaces to conform to KIP17 via KIP13
        _registerInterface(_INTERFACE_ID_KIP7BURNABLE);
    }

    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See `KIP7._burn`.
     */
    function burn(uint256 amount) public {
        _burn(msg.sender, amount);
    }

    /**
     * @dev See `KIP7._burnFrom`.
     */
    function burnFrom(address account, uint256 amount) public {
        _burnFrom(account, amount);
    }
}
```

```
contract KIP7Metadata is KIP13, IKIP7 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    /*
     *     bytes4(keccak256('name()')) == 0x06fdde03
     *     bytes4(keccak256('symbol()')) == 0x95d89b41
     *     bytes4(keccak256('decimals()')) == 0x313ce567
     *
     *     => 0x06fdde03 ^ 0x95d89b41 ^ 0x313ce567 == 0xa219a025
     */
    bytes4 private constant _INTERFACE_ID_KIP7_METADATA = 0xa219a025;

    /**
     * @dev Sets the values for `name`, `symbol`, and `decimals`. All three of
     * these values are immutable: they can only be set once during
     * construction.
     */
    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;

        // register the supported interfaces to conform to KIP7 via KIP13
        _registerInterface(_INTERFACE_ID_KIP7_METADATA);
    }

    /**
     * @dev Returns the name of the token.
     */
    function name() public view returns (string memory) {
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view returns (string memory) {
        return _symbol;
```

```
    }

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei.
     *
     * > Note that this information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * `IKIP7.balanceOf` and `IKIP7.transfer`.
     */
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}

contract KIP7Mintable is KIP13, KIP7, MinterRole {
    /*
     *       bytes4(keccak256('mint(address,uint256)')) == 0x40c10f19
     *       bytes4(keccak256('isMinter(address)')) == 0xaa271e1a
     *       bytes4(keccak256('addMinter(address)')) == 0x983b2d56
     *       bytes4(keccak256('renounceMinter()')) == 0x98650275
     *
     *       => 0x40c10f19 ^ 0xaa271e1a ^ 0x983b2d56 ^ 0x98650275 == 0xeab83e20
     */
    bytes4 private constant _INTERFACE_ID_KIP7MINTABLE = 0xeab83e20;

    constructor () public {
        // register the supported interfaces to conform to KIP17 via KIP13
        _registerInterface(_INTERFACE_ID_KIP7MINTABLE);
    }

    /**
     * @dev See `KIP7._mint`.
     *
     * Requirements:
     *
     * - the caller must have the `MinterRole`.
     */
```

```
    function mint(address account, uint256 amount) public onlyMinter returns (bool) {
        _mint(account, amount);
        return true;
    }
}

contract KIP7Pausable is KIP13, KIP7, Pausable {
    /*
     *     bytes4(keccak256('paused()')) == 0x5c975abb
     *     bytes4(keccak256('pause()')) == 0x8456cb59
     *     bytes4(keccak256('unpause()')) == 0x3f4ba83a
     *     bytes4(keccak256('isPauser(address)')) == 0x46fbf68e
     *     bytes4(keccak256('addPauser(address)')) == 0x82dc1ec4
     *     bytes4(keccak256('renouncePauser()')) == 0x6ef8d66d
     *
     *     => 0x5c975abb ^ 0x8456cb59 ^ 0x3f4ba83a ^ 0x46fbf68e ^ 0x82dc1ec4 ^ 0x6ef8d66d == 0x4d5507ff
     */
    bytes4 private constant _INTERFACE_ID_KIP7PAUSABLE = 0x4d5507ff;

    constructor () public {
        // register the supported interfaces to conform to KIP17 via KIP13
        _registerInterface(_INTERFACE_ID_KIP7PAUSABLE);
    }

    function transfer(address to, uint256 value) public whenNotPaused returns (bool) {
        return super.transfer(to, value);
    }

    function transferFrom(address from, address to, uint256 value) public whenNotPaused returns (bool) {
        return super.transferFrom(from, to, value);
    }

    function approve(address spender, uint256 value) public whenNotPaused returns (bool) {
        return super.approve(spender, value);
    }
}

contract KIP7Token is KIP7Mintable, KIP7Burnable, KIP7Pausable, KIP7Metadata {
    constructor(string memory name, string memory symbol, uint8 decimals, uint256 initialSupply) KIP7Metadata(name,
symbol, decimals) public {
        _mint(msg.sender, initialSupply);
    }
```

```
}

library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * This test is non-exhaustive, and there may be false-negatives: during the
     * execution of a contract's constructor, its address will be reported as
     * not containing a contract.
     *
     * > It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }
}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account