Smart Contract Security Audit Report

The SlowMist Security Team received the Ultra team's application for smart contract security audit of the UltraToken on November 13, 2019. The following are the details and results of this smart contract security audit:

Token name :

UltraToken

The Contract address :

0xD13c7342e1ef687C5ad21b27c2b65D772cAb5C8c

Link address :

https://etherscan.io/address/0xD13c7342e1ef687C5ad21b27c2b65D772cAb5C8c#code

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 : 0X001911130001

Audit Date : November 13, 2019

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 contains the tokenVault section. SafeMath security module is used, which is a commendable approach. The contract does not have the Overflow and the Race Conditions issue. The comprehensive evaluation contract is no risk.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2019-07-17
*/

// File: openzeppelin-solidity/contracts/token/ERC20/IERC20.sol

//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity ^0.5.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see `ERC20Detailed`.
```

```
 */
interface IERC20 {
    /**
     * @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 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);
}

// File: openzeppelin-solidity/contracts/math/SafeMath.sol

pragma solidity ^0.5.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
```

```
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
```

//SlowMist// 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-solidity/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;
    }
}

// File: openzeppelin-solidity/contracts/token/ERC20/ERC20.sol

pragma solidity ^0.5.0;



/**
 * @dev Implementation of the `IERC20` interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using `_mint`.
 * For a generic mechanism see `ERC20Mintable`.
 *
 * *For a detailed writeup see our guide [How to implement supply
 * mechanisms](https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226).*
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an `Approval` event is emitted on calls to `transferFrom`.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard `decreaseAllowance` and `increaseAllowance`
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See `IERC20.approve`.
 */
contract ERC20 is IERC20 {
    using SafeMath for uint256;
```

```
mapping (address => uint256) private _balances;

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

uint256 private _totalSupply;

/**
 * @dev See `IERC20.totalSupply`.
 */
function totalSupply() public view returns (uint256) {
    return _totalSupply;
}

/**
 * @dev See `IERC20.balanceOf`.
 */
function balanceOf(address account) public view returns (uint256) {
    return _balances[account];
}

/**
 * @dev See `IERC20.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 EIP20 specification
}

/**
 * @dev See `IERC20.allowance`.
 */
function allowance(address owner, address spender) public view returns (uint256) {
    return _allowances[owner][spender];
}

/**
 * @dev See `IERC20.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 EIP20 specification
}

/**
 * @dev See `IERC20.transferFrom`.
 *
 * Emits an `Approval` event indicating the updated allowance. This is not
 * required by the EIP. See the note at the beginning of `ERC20`;
 *
 * 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 EIP20 specification
}

/**
 * @dev Atomically increases the allowance granted to `spender` by the caller.
 *
 * This is an alternative to `approve` that can be used as a mitigation for
 * problems described in `IERC20.approve`.
 *
 * Emits an `Approval` event indicating the updated allowance.
 *
 * Requirements:
 *
 * - `spender` cannot be the zero address.
 */
function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
    _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue));
    return true;    }
```

```
    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to `approve` that can be used as a mitigation for
     * problems described in `IERC20.approve`.
     *
     * Emits an `Approval` event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(subtractedValue));
        return true;
    }

    /**
     * @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), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: 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.
 */
 //SlowMist// mint function
function _mint(address account, uint256 amount) internal {
    require(account != address(0), "ERC20: mint to the zero address");

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

 /**
 * @dev Destoys `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), "ERC20: 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 is 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), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

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

    /**
     * @dev Destoys `amount` tokens from `account`.`amount` is then deducted
     * from the caller's allowance.
     *
     * See `_burn` and `_approve`.
     */
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount));
    }
}

// File: openzeppelin-solidity/contracts/access/Roles.sol

pragma solidity ^0.5.0;

/**
 * @title Roles
 * @dev Library for managing addresses assigned to a Role.
 */
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];
    }
}

// File: openzeppelin-solidity/contracts/access/roles/PauserRole.sol

pragma solidity ^0.5.0;


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);
    }
}

// File: openzeppelin-solidity/contracts/lifecycle/Pausable.sol

pragma solidity ^0.5.0;


/**
 * @dev Contract module which allows children to implement an emergency stop
 * mechanism that can be triggered by an authorized account.
 *
 * This module is used through inheritance. It will make available the
 * modifiers `whenNotPaused` and `whenPaused`, which can be applied to
 * the functions of your contract. Note that they will not be pausable by
 * simply including this module, only once the modifiers are put in place.
 */
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);
    }
}

// File: openzeppelin-solidity/contracts/token/ERC20/ERC20Pausable.sol

pragma solidity ^0.5.0;



/**
 * @title Pausable token
 * @dev ERC20 modified with pausable transfers.
 */
contract ERC20Pausable is ERC20, Pausable {
    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);
    }

    function increaseAllowance(address spender, uint addedValue) public whenNotPaused returns (bool) {
        return super.increaseAllowance(spender, addedValue);
    }
```

```
    function decreaseAllowance(address spender, uint subtractedValue) public whenNotPaused returns (bool) {
        return super.decreaseAllowance(spender, subtractedValue);
    }
}

// File: openzeppelin-solidity/contracts/ownership/Ownable.sol

pragma solidity ^0.5.0;

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be aplied to your functions to restrict their use to
 * the owner.
 */
contract Ownable {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
```

```
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * > Note: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address"); //SlowMist// This check is quite good in avoiding losing control of the contract caused by user mistakes
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

// File: contracts/Ultra_IEO.sol
```

```
pragma solidity 0.5.9;

//import 'openzeppelin-solidity/contracts/math/Math.sol';



contract UltraToken is ERC20Pausable, Ownable {
    using SafeMath for uint256;
//  using Math    for uint256;

    string  private _name = "Ultra Token";
    string  private _symbol = "UOS";
    uint8   private _decimals = 4;      // the token precision on UOS mainnet is 4, can be adjusted there as necessary.

    /*  0-------------------------------------->  time in month(30 days)
     *  ^             ^                 ^
     *  |             |                 |
     * deploy        start     ...       end
     *              10.00%    ...      20.00%    adds up to 100.00%
     */
    uint256 private _deployTime;              // the deployment time of the contract
    uint256 private _month = 30 days;          // time unit
    struct VestingContract {
        uint256[]   basisPoints;    // the basis points array of each vesting. The last one won't matter, cause we give the remainder to the user at last, but the sum will be validated against 10000
        uint256     startMonth;     // the index of the month at the beginning of which the first vesting is available
        uint256     endMonth;       // the index of the month at the beginning of which the last vesting is available; _endMonth = _startMonth + _basisPoints.length -1;
    }

    struct BuyerInfo {
        uint256 total;          // the total number of tokens purchased by the buyer
        uint256 claimed;        // the number of tokens has been claimed so far
        string  contractName;   // with what contract the buyer purchased the tokens
    }

    mapping (string => VestingContract) private _vestingContracts;
    mapping (address => BuyerInfo)      private _buyerInfos;

    mapping (address => string) private _keys;              // ethereum to eos public key mapping
    mapping (address => bool)   private _updateApproval;    // whether allow an account to update its registerred
```

```
eos key

    constructor() public {
        _mint(address(this), uint256(1000000000).mul(10**uint256(_decimals)));  // mint all tokens and send to
the contract, 1,000,000,000 UOS;
        _deployTime = block.timestamp;
    }

    function name() public view returns (string memory) {
        return _name;
    }

    function symbol() public view returns (string memory) {
        return _symbol;
    }

    function decimals() public view returns (uint8) {
        return _decimals;
    }

    function deployTime() public view returns (uint256) {
        return _deployTime;
    }

    function buyerInformation() public view returns (uint256, uint256, string memory) {
        BuyerInfo storage buyerInfo = _buyerInfos[msg.sender];
        return (buyerInfo.total, buyerInfo.claimed, buyerInfo.contractName );
    }

    function nextVestingDate() public view returns (uint256) {
        BuyerInfo storage buyerInfo = _buyerInfos[msg.sender];
        require(buyerInfo.total > 0, "Buyer does not exist");
        VestingContract storage vestingContract = _vestingContracts[buyerInfo.contractName];
        uint256 currentMonth = block.timestamp.sub(_deployTime).div(_month);
        if(currentMonth < vestingContract.startMonth) {
            return _deployTime.add(vestingContract.startMonth.mul(_month));
        } else if(currentMonth >= vestingContract.endMonth) {
            return _deployTime.add(vestingContract.endMonth.mul(_month));
        } else {
            return _deployTime.add(currentMonth.add(1).mul(_month));
        }
    }
```

```
    event SetVestingContract(string contractName, uint256[] basisPoints, uint256 startMonth);

    function setVestingContract(string memory contractName, uint256[] memory basisPoints, uint256 startMonth)
public onlyOwner whenNotPaused returns (bool) {
        VestingContract storage vestingContract = _vestingContracts[contractName];
        require(vestingContract.basisPoints.length == 0, "can't change an existing contract");
        uint256 totalBPs = 0;
        for(uint256 i = 0; i < basisPoints.length; i++) {
            totalBPs = totalBPs.add(basisPoints[i]);
        }
        require(totalBPs == 10000, "invalid basis points array"); // this also ensures array length is not zero

        vestingContract.basisPoints = basisPoints;
        vestingContract.startMonth  = startMonth;
        vestingContract.endMonth    = startMonth.add(basisPoints.length).sub(1);

        emit SetVestingContract(contractName, basisPoints, startMonth);
        return true;
    }

    event ImportBalance(address[] buyers, uint256[] tokens, string contractName);

    // import balance for a group of user with a specific contract terms
    function importBalance(address[] memory buyers, uint256[] memory tokens, string memory contractName) public
onlyOwner whenNotPaused returns (bool) {
        require(buyers.length == tokens.length, "buyers and balances mismatch");

        VestingContract storage vestingContract = _vestingContracts[contractName];
        require(vestingContract.basisPoints.length > 0, "contract does not exist");

        for(uint256 i = 0; i < buyers.length; i++) {
            require(tokens[i] > 0, "cannot import zero balance");
            BuyerInfo storage buyerInfo = _buyerInfos[buyers[i]];
            require(buyerInfo.total == 0, "have already imported balance for this buyer");
            buyerInfo.total = tokens[i];
            buyerInfo.contractName = contractName;
        }

        emit ImportBalance(buyers, tokens, contractName);
        return true;
    }

    event Claim(address indexed claimer, uint256 claimed);
```

```
    function claim() public whenNotPaused returns (bool) {
        uint256 canClaim = claimableToken();

        require(canClaim > 0, "No token is available to claim");

        _buyerInfos[msg.sender].claimed = _buyerInfos[msg.sender].claimed.add(canClaim);
        _transfer(address(this), msg.sender, canClaim);

        emit Claim(msg.sender, canClaim);
        return true;
    }

    // the number of token can be claimed by the msg.sender at the moment
    function claimableToken() public view returns (uint256) {
        BuyerInfo storage buyerInfo = _buyerInfos[msg.sender];

        if(buyerInfo.claimed < buyerInfo.total) {
            VestingContract storage vestingContract = _vestingContracts[buyerInfo.contractName];
            uint256 currentMonth = block.timestamp.sub(_deployTime).div(_month);

            if(currentMonth < vestingContract.startMonth) {
                return uint256(0);
            }

            if(currentMonth >= vestingContract.endMonth) { // vest the unclaimed token all at once so there's
no remainder
                return buyerInfo.total.sub(buyerInfo.claimed);
            } else {
                uint256 claimableIndex = currentMonth.sub(vestingContract.startMonth);
                uint256 canClaim = 0;
                for(uint256 i = 0; i <= claimableIndex; ++i) {
                    canClaim = canClaim.add(vestingContract.basisPoints[i]);
                }
                return canClaim.mul(buyerInfo.total).div(10000).sub(buyerInfo.claimed);
            }
        }
        return uint256(0);
    }

    event SetKey(address indexed buyer, string EOSKey);

    function _register(string memory EOSKey) internal {
```

```
        require(bytes(EOSKey).length > 0 && bytes(EOSKey).length <= 64, "EOS public key length should be less than 64 characters");
        _keys[msg.sender] = EOSKey;

        emit SetKey(msg.sender, EOSKey);
    }

    function register(string memory EOSKey) public whenNotPaused returns (bool) {
        _register(EOSKey);
        return true;
    }

    function keyOf() public view returns (string memory) {
        return _keys[msg.sender];
    }

    event SetUpdateApproval(address indexed buyer, bool isApproved);

    function setUpdateApproval(address buyer, bool isApproved) public onlyOwner returns (bool) {
        require(balanceOf(buyer) > 0 || _buyerInfos[buyer].total > 0, "This account has no token"); // allowance will not be considered
        _updateApproval[buyer] = isApproved;

        emit SetUpdateApproval(buyer, isApproved);
        return true;
    }

    function updateApproved() public view returns (bool) {
        return _updateApproval[msg.sender];
    }

    function update(string memory EOSKey) public returns (bool) {
        require(_updateApproval[msg.sender], "Need approval from ultra after contract is frozen");
        _register(EOSKey);
        return true;
    }

}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account