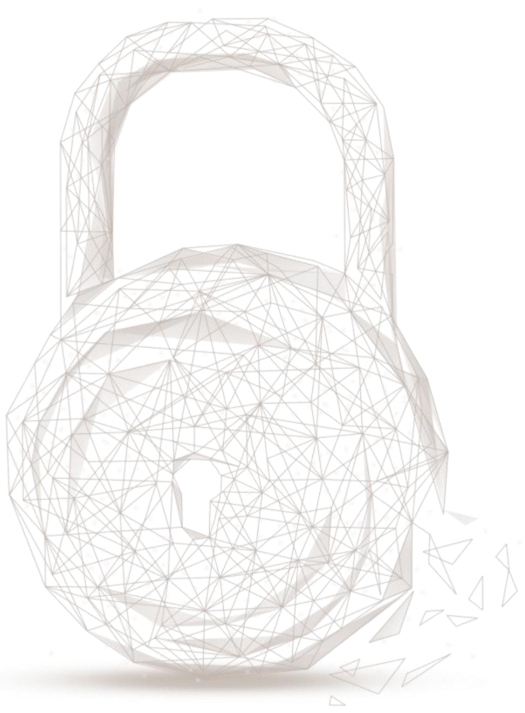
BEOSIN

Blockchain Security

Smart contract security audit report



Audit Number：202101181100

Smart Contract Name：

Lavaswap (Lava)

Smart Contract Address：

0x56f95662e71f30b333b456439248c6de589082a4

Smart Contract Address Link：

https://scan.hecochain.com/address/0x56f95662e71f30b333b456439248c6de589082a4#contracts

Start Date：2021.01.18

Completion Date：2021.01.18

Overall Result：Pass（Distinction）

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	HRC20 Token Standards	Pass
		Compiler Version Security	Pass
		Visibility Specifiers	Pass
		Gas Consumption	Pass
		SafeMath Features	Pass
		Fallback Usage	Pass
		tx.origin Usage	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		Overriding Variables	Pass
2	Function Call Audit	Authorization of Function Call	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		selfdestruct Function Security	Pass

3	Business Security	Access Control of Owner	Pass
		Business Logics	Pass
		Business Implementations	Pass
4	Integer Overflow/Underflow	-	Pass
5	Reentrancy	-	Pass
6	Exceptional Reachable State	-	Pass
7	Transaction-Ordering Dependence	-	Pass
8	Block Properties Dependence	-	Pass
9	Pseudo-random Number Generator (PRNG)	-	Pass
10	DoS (Denial of Service)	-	Pass
11	Token Vesting Implementation	-	N/A
12	Fake Deposit	-	Pass
13	event security	-	Pass

Note: Audit results and suggestions in code comments

Disclaimer: This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract Lava, including Coding Standards, Security, and Business Logic. **Lava contract passed all audit items. The overall result is Pass (Distinction). The smart contract is able to function properly.** Please find below the basic information of the smart contract:

1. Basic Token Information

Token name	Lavaswap
Token symbol	Lava
decimals	18
totalSupply	Initial supply is 100 million(Mintable with a cap of 200 million, burnable)
Token type	HRC20

Table 1 – Basic Token Information

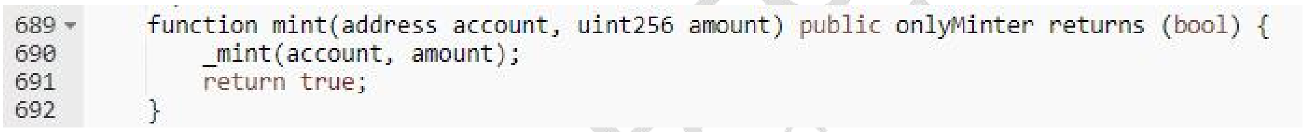
2. Token Vesting Information

N/A

3. Other function description

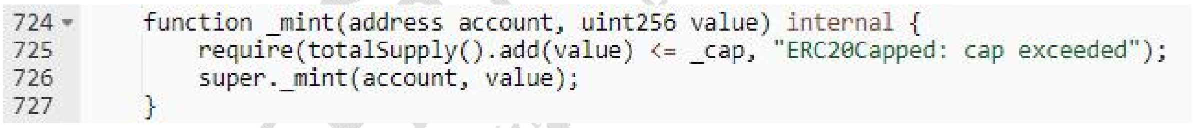
(1) mint/burn function

As shown in the Figure below, the contract implements *mint, burn & burnFrom* functions to mint & burn tokens. The mint cap of Lava contracts is 200 million, and the initial supply is 100 million, which cannot be changed after deployment. Only minter can call the *mint* function to mint tokens, the default contract creator is the minter. Only minters can call the *addMinter* function to add new minters, and call the *renounceMinter* function to delete the caller's own mint permissions, and the total supply of tokens cannot exceed 200 million.



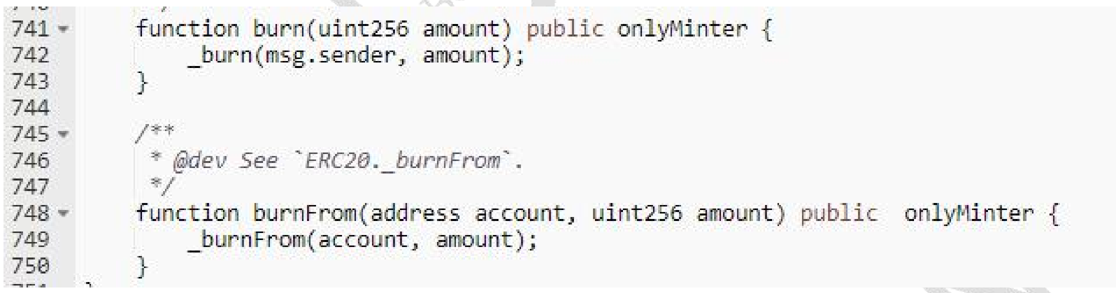
```
    function mint(address account, uint256 amount) public onlyMinter returns (bool) {
        _mint(account, amount);
        return true;
    }
```

Figure 1 mint Function Source Code



```
    function _mint(address account, uint256 value) internal {
        require(totalSupply().add(value) <= _cap, "ERC20Capped: cap exceeded");
        super._mint(account, value);
    }
```

Figure 2 _mint Function Source Code



```
    function burn(uint256 amount) public onlyMinter {
        _burn(msg.sender, amount);
    }

    /**
     * @dev See `ERC20._burnFrom`.
     */
    function burnFrom(address account, uint256 amount) public  onlyMinter {
        _burnFrom(account, amount);
    }
```

Figure 3 burn/burnFrom Function Source Code

Audited Source Code with Comments:

```
/**
 *Submitted for verification at Etherscan.io on 2019-06-05
*/

pragma solidity ^0.5.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.
// Beosin (Chengdu LianAn) // Define the function interfaces required by HRC20 Token standard.

/**
 * @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);
}

/**
 * @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.
```

```
  */
// Beosin (Chengdu LianAn) // The SafeMath contract declares four functions for safe mathematical
operation.
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;
    }
}
```

```
/**
 * @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; // Beosin (Chengdu LianAn) // Use the SafeMath library for
mathematical operation. Avoid integer overflow/underflow.

    mapping (address => uint256) private _balances; // Beosin (Chengdu LianAn) // Declare the mapping
variable '_balances' for storing the token balance of specified address.

    mapping (address => mapping (address => uint256)) private _allowances; // Beosin (Chengdu LianAn)
// Declare the mapping variable '_allowances' for storing the allowance between specified addresses.

    uint256 private _totalSupply; // Beosin (Chengdu LianAn) // Declare the variable '_totalSupply' for
storing the total token supply.

    /**
     * @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`.
     */
    // Beosin (Chengdu LianAn) // The 'transfer' function, function caller sends a certain amount of
tokens to the destination address 'recipient'.
    function transfer(address recipient, uint256 amount) public returns (bool) {
        _transfer(msg.sender, recipient, amount);
        return true;
    }

    /**
     * @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.
     */
    // Beosin (Chengdu LianAn) // 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.
    // Beosin (Chengdu LianAn) // Using function 'increaseAllowance' and 'decreaseAllowance' to alter
allowance is recommended.
    function approve(address spender, uint256 value) public returns (bool) {
        _approve(msg.sender, spender, value); // Beosin (Chengdu LianAn) // Call the internal function
'_approve' to set the allowance.
        return true;
    }

    /**
     * @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`.
     */
    // Beosin (Chengdu LianAn) // The 'transferFrom' function, 'msg.sender' as a delegate of 'sender'
transfers the specified amount of tokens to a specified address.
    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;
    }

    /**
     * @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)); // Beosin
(Chengdu LianAn) // Call the internal function '_approve' to increase the allowance.
        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)); // Beosin
(Chengdu LianAn) // Call the internal function '_approve' to decrease the allowance.
        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"); // Beosin (Chengdu
LianAn) // Non-zero address check for the address 'sender'.
        require(recipient != address(0), "ERC20: transfer to the zero address"); // Beosin (Chengdu
LianAn) // Non-zero address check for the address 'recipient'.

        _balances[sender] = _balances[sender].sub(amount); // Beosin (Chengdu LianAn) // Alter the
token balance of 'sender'.
        _balances[recipient] = _balances[recipient].add(amount); // Beosin (Chengdu LianAn) // Alter the
token balance of 'recipient'.
        emit Transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Trigger the event
'Transfer'.
    }

    /** @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), "ERC20: mint to the zero address"); // Beosin (Chengdu LianAn) //
Non-zero address check for the address 'account'.

        _totalSupply = _totalSupply.add(amount); // Beosin (Chengdu LianAn) // Update the total supply
of token.
        _balances[account] = _balances[account].add(amount); // Beosin (Chengdu LianAn) // Alter the
token balance of 'account'.
        emit Transfer(address(0), account, amount); // Beosin (Chengdu LianAn) // Trigger the event
```

```
'Transfer'.
        }

          /**
          * @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"); // Beosin (Chengdu LianAn)
// Non-zero address check for the address 'account'.

                _totalSupply = _totalSupply.sub(value); // Beosin (Chengdu LianAn) // Update the total supply
of token.
                _balances[account] = _balances[account].sub(value); // Beosin (Chengdu LianAn) // Alter the
token balance of 'account'.
                emit Transfer(account, address(0), value); // Beosin (Chengdu LianAn) // Trigger the event
'Transfer'.
        }

        /**
          * @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"); // Beosin (Chengdu
LianAn) // Non-zero address check for the address 'owner'.
                require(spender != address(0), "ERC20: approve to the zero address"); // Beosin (Chengdu
LianAn) // Non-zero address check for the address 'spender'.

                _allowances[owner][spender] = value; // Beosin (Chengdu LianAn) // The allowance which
address 'owner' allowed to 'spender' is set to 'value'.
                emit Approval(owner, spender, value); // Beosin (Chengdu LianAn) // Trigger the event
```

```
'Approval'.
        }

        /**
         * @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); // Beosin (Chengdu LianAn) // Call the internal function '_burn' to
destroy tokens.
                _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount)); // Beosin
(Chengdu LianAn) // Call the internal function '_approve' to update the allowance.
        }
}

/**
 * @dev Optional functions from the ERC20 standard.
 */
contract ERC20Detailed is IERC20 {
        string private _name; // Beosin (Chengdu LianAn) // Declare the variable '_name' for storing the
name of token.
        string private _symbol; // Beosin (Chengdu LianAn) // Declare the variable '_symbol' for storing the
symbol of token.
        uint8 private _decimals; // Beosin (Chengdu LianAn) // Declare the variable '_decimals' for storing
the decimals of token.

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

        /**
         * @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
     * `IERC20.balanceOf` and `IERC20.transfer`.
     */
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}

/**
 * @title Roles
 * @dev Library for managing addresses assigned to a Role.
 */
library Roles {
    // Beosin (Chengdu LianAn) // Declare structure 'Role' for storing whether the address has corresponding permission.
    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"); // Beosin (Chengdu LianAn) // Require that address 'account' is not the zero address and does not have corresponding permission.
        role.bearer[account] = true; // Beosin (Chengdu LianAn) // Grant corresponding permission to the address 'account'.
    }

    /**
     * @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"); // Beosin (Chengdu LianAn) //
Require that address 'account' is not the zero address and has corresponding permission.
                role.bearer[account] = false; // Beosin (Chengdu LianAn) // Remove corresponding permission
for address 'account'.
        }

        /**
          * @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"); // Beosin (Chengdu LianAn)
// Require that address 'account' is not the zero address and has corresponding permission.
                return role.bearer[account]; // Beosin (Chengdu LianAn) // Return whether the address
'account' has corresponding permission.
        }
}

contract PauserRole {
        using Roles for Roles.Role; // Beosin (Chengdu LianAn) // Attach the functions from the library
'Roles' to data type 'Roles.Role'.
        // Beosin (Chengdu LianAn) // Declare the events 'PauserAdded' and 'PauserRemoved'.
        event PauserAdded(address indexed account);
        event PauserRemoved(address indexed account);

        Roles.Role private _pausers; // Beosin (Chengdu LianAn) // Define the permission '_pausers'.

        constructor () internal {
                _addPauser(msg.sender); // Beosin (Chengdu LianAn) // Grant permission '_pausers' to the
address of deploying this contract.
        }

        modifier onlyPauser() {
                require(isPauser(msg.sender), "PauserRole: caller does not have the Pauser role"); // Beosin
(Chengdu LianAn) // Require that the function caller must be have the permission '_pausers'.
                _;
        }
        // Beosin (Chengdu LianAn) // Check whether the specified address has the permission '_pausers'.
        function isPauser(address account) public view returns (bool) {
                return _pausers.has(account);
        }
        // Beosin (Chengdu LianAn) // Grant permission '_pausers' to the specified address.
        function addPauser(address account) public onlyPauser {
                _addPauser(account);
        }
        // Beosin (Chengdu LianAn) // Renounce the permission '_pausers'.
        function renouncePauser() public {
```

```
        _removePauser(msg.sender);
    }
    // Beosin (Chengdu LianAn) // The basic implementation function that grants the permission
'_pausers' to specified address.
    function _addPauser(address account) internal {
        _pausers.add(account); // Beosin (Chengdu LianAn) // Grant permission '_pausers' to 'account'.
        emit PauserAdded(account); // Beosin (Chengdu LianAn) // Trigger the event 'PauserAdded'.
    }
    // Beosin (Chengdu LianAn) // The basic implementation function that removes the permission
'_pausers' for the specified address.
    function _removePauser(address account) internal {
        _pausers.remove(account); // Beosin (Chengdu LianAn) // Remove the permission '_pausers' for
'account'.
        emit PauserRemoved(account); // Beosin (Chengdu LianAn) // Trigger the event
'PauserRemoved'.
    }
}

/**
 * @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); // Beosin (Chengdu LianAn) // Declare the event 'Paused'.

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

    bool private _paused; // Beosin (Chengdu LianAn) // Declare the variable '_paused' for storing the
pause status.

    /**
     * @dev Initializes the contract in unpaused state. Assigns the Pauser role
     * to the deployer.
     */
    // Beosin (Chengdu LianAn) // Constructor, initialize the pause status to 'false'.
    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.
     */
    function pause() public onlyPauser whenNotPaused {
        _paused = true; // Beosin (Chengdu LianAn) // Change the pause status to 'true'.
        emit Paused(msg.sender); // Beosin (Chengdu LianAn) // Trigger the event 'Paused'.
    }

    /**
     * @dev Called by a pauser to unpause, returns to normal state.
     */
    function unpause() public onlyPauser whenPaused {
        _paused = false; // Beosin (Chengdu LianAn) // Change the pause status to 'false'.
        emit Unpaused(msg.sender); // Beosin (Chengdu LianAn) // Trigger the event 'Unpaused'.
    }
}

/**
 * @title Pausable token
 * @dev ERC20 modified with pausable transfers.
 */
contract ERC20Pausable is ERC20, Pausable {
    // Beosin (Chengdu LianAn) // Rewrite the function 'transfer', add the modifier 'whenNotPaused',
the address with '_pausers' permission can pause the function for transferring tokens.
    function transfer(address to, uint256 value) public whenNotPaused returns (bool) {
        return super.transfer(to, value);
    }
    // Beosin (Chengdu LianAn) // Rewrite the function 'transferFrom', add the modifier
'whenNotPaused', the address with '_pausers' permission can pause the function for delegate
transferring tokens.
    function transferFrom(address from, address to, uint256 value) public whenNotPaused returns (bool) {
        return super.transferFrom(from, to, value);
    }
    // Beosin (Chengdu LianAn) // Rewrite the function 'approve', add the modifier 'whenNotPaused',
the address with '_pausers' permission can pause the function for setting the allowance.
    function approve(address spender, uint256 value) public whenNotPaused returns (bool) {
        return super.approve(spender, value);
    }
    // Beosin (Chengdu LianAn) // Rewrite the function 'increaseAllowance', add the modifier
'whenNotPaused', the address with '_pausers' permission can pause the function for increasing the
allowance.
    function increaseAllowance(address spender, uint addedValue) public whenNotPaused returns (bool) {
        return super.increaseAllowance(spender, addedValue);
    }
    // Beosin (Chengdu LianAn) // Rewrite the function 'decreaseAllowance', add the modifier
'whenNotPaused', the address with '_pausers' permission can pause the function for decreasing the
allowance.
    function decreaseAllowance(address spender, uint subtractedValue) public whenNotPaused returns
(bool) {
        return super.decreaseAllowance(spender, subtractedValue);
    }
}

contract MinterRole {
        using Roles for Roles.Role; // Beosin (Chengdu LianAn) // Attach functions in Roles library to
data type 'Roles.Role'.

    event MinterAdded(address indexed account); // Beosin (Chengdu LianAn) // Declare the event
'MinterAdded'.
    event MinterRemoved(address indexed account); // Beosin (Chengdu LianAn) // Declare the event
'MinterRemoved'.

    Roles.Role private _minters; // Beosin (Chengdu LianAn) // Declare permission '_minters'.

    constructor () internal {
        _addMinter(msg.sender); // Beosin (Chengdu LianAn) // Call internal function '_addMinter' to
grant contract deployer permission '_minters'.
    }
    // Beosin (Chengdu LianAn) // Modifier, require the modified function caller must be have
permission '_minters'.
    modifier onlyMinter() {
        require(isMinter(msg.sender), "MinterRole: caller does not have the Minter role");
        _;
    }
    // Beosin (Chengdu LianAn) // Check whether the specified address has permission '_minters'.
    function isMinter(address account) public view returns (bool) {
        return _minters.has(account);
    }
    // Beosin (Chengdu LianAn) // Grant the permission '_minters' to the specified address.
    function addMinter(address account) public onlyMinter {
        _addMinter(account); // Beosin (Chengdu LianAn) // Call internal function '_addMinter' to
grant the permission '_minters' to address 'account'.
    }
    // Beosin (Chengdu LianAn) // Renounce the permission '_minters'.
    function renounceMinter() public {
        _removeMinter(msg.sender); // Beosin (Chengdu LianAn) // Call internal function
'_removeMinter' to remove the permission '_minters' for function caller.
    }
    // Beosin (Chengdu LianAn) // The basic function granting permission '_minters'.
    function _addMinter(address account) internal {
        _minters.add(account); // Beosin (Chengdu LianAn) // Call internal function 'add' to grant the
permission '_minters' to address 'account'.
        emit MinterAdded(account); // Beosin (Chengdu LianAn) // Trigger the event 'MinterAdded'.
    }
    // Beosin (Chengdu LianAn) // The basic function of removing permission '_minters'.
    function _removeMinter(address account) internal {
        _minters.remove(account); // Beosin (Chengdu LianAn) // Call internal function 'remove' to
remove the permission '_minters' for address 'account'.
        emit MinterRemoved(account); // Beosin (Chengdu LianAn) // Trigger the event
'MinterRemoved'.
    }
}

/**
 * @dev Extension of `ERC20` that adds a set of accounts with the `MinterRole`,
 * which have permission to mint (create) new tokens as they see fit.
 *
 * At construction, the deployer of the contract is the only minter.
 */
contract ERC20Mintable is ERC20, MinterRole {
    /**
     * @dev See `ERC20._mint`.
     *
     * Requirements:
     *
     * - the caller must have the `MinterRole`.
     */
    // Beosin (Chengdu LianAn) // Mint tokens to specified address.
    function mint(address account, uint256 amount) public onlyMinter returns (bool) {
        _mint(account, amount); // Beosin (Chengdu LianAn) // Call the internal function '_mint' to
mint tokens.
        return true;
    }
}

/**
 * @dev Extension of `ERC20Mintable` that adds a cap to the supply of tokens.
 */
contract ERC20Capped is ERC20Mintable {
    uint256 private _cap; // Beosin (Chengdu LianAn) // Declare variable '_cap' for storing the cap of
token.

    /**
     * @dev Sets the value of the `cap`. This value is immutable, it can only be
     * set once during construction.
     */
    constructor (uint256 cap) public {
        require(cap > 0, "ERC20Capped: cap is 0"); // Beosin (Chengdu LianAn) // Require 'cap' must be
greater than 0.
        _cap = cap; // Beosin (Chengdu LianAn) // Set the cap of token.
    }

    /**
     * @dev Returns the cap on the token's total supply.
     */
    function cap() public view returns (uint256) {
        return _cap;
    }

    /**
     * @dev See `ERC20Mintable.mint`.
     *
     * Requirements:
     *
     * - `value` must not cause the total supply to go over the cap.
     */
    // Beosin (Chengdu LianAn) // Rewrite the basic function of minting tokens, add the cap check.
    function _mint(address account, uint256 value) internal {
        require(totalSupply().add(value) <= _cap, "ERC20Capped: cap exceeded"); // Beosin (Chengdu
LianAn) // Require that the total amount of tokens after this minting does not exceed the cap.
        super._mint(account, value); // Beosin (Chengdu LianAn) // Call the function '_mint' of the
parent contract to mint tokens.
    }
}

/**
 * @dev Extension of `ERC20` that allows token holders to destroy both their own
 * tokens and those that they have an allowance for, in a way that can be
 * recognized off-chain (via event analysis).
 */
contract ERC20Burnable is ERC20, MinterRole {
    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See `ERC20._burn`.
     */
    function burn(uint256 amount) public onlyMinter {
        _burn(msg.sender, amount); // Beosin (Chengdu LianAn) // Call the internal function '_burn' to
destroy tokens.
    }

    /**
     * @dev See `ERC20._burnFrom`.
     */
    function burnFrom(address account, uint256 amount) public   onlyMinter {
        _burnFrom(account, amount); // Beosin (Chengdu LianAn) // Call the internal function
'_burnFrom' to burn tokens.
    }
}

/**
 * @title SKM Protocol ERC20 Token Contract
 * @author
 *
 * @dev Implementation of the New SKM Token.
 */
contract Lavaswap is ERC20Detailed, ERC20Capped, ERC20Burnable, ERC20Pausable {
    string   private constant   TOKEN_NAME       = "Lavaswap"; // Beosin (Chengdu LianAn) //
Declare the constant 'TOKEN_NAME' for storing the token name, it is default 'Lavaswap'.
    string   private constant TOKEN_SYMBOL     = "Lava"; // Beosin (Chengdu LianAn) // Declare the
constant 'TOKEN_SYMBOL' for storing the token symbol, it is default 'Lava'.
    uint private constant INITIAL_TOKENS = 100000000; // Beosin (Chengdu LianAn) // Declare the
constant 'INITIAL_TOKENS' for storing the initial total token supply, it is default 100 million.
    uint8 private constant DECIMALS = 18; // Beosin (Chengdu LianAn) // Declare constant
'DECIMALS' for storing the token decimals, it is default 18.
    uint256 public constant INITIAL_SUPPLY = INITIAL_TOKENS * (10 ** uint256(DECIMALS));
    // Beosin (Chengdu LianAn) // Constructor, initialize the basic token information and mint cap.
    constructor()   public
        ERC20Detailed(TOKEN_NAME,TOKEN_SYMBOL,DECIMALS)
        ERC20Capped(INITIAL_SUPPLY*2)
    {
        _mint(msg.sender, INITIAL_SUPPLY); // Beosin (Chengdu LianAn) // Call internal function
'_mint' to send all initial tokens to contract deployer.
    }
}
```

Beosin

BEOSIN

Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com