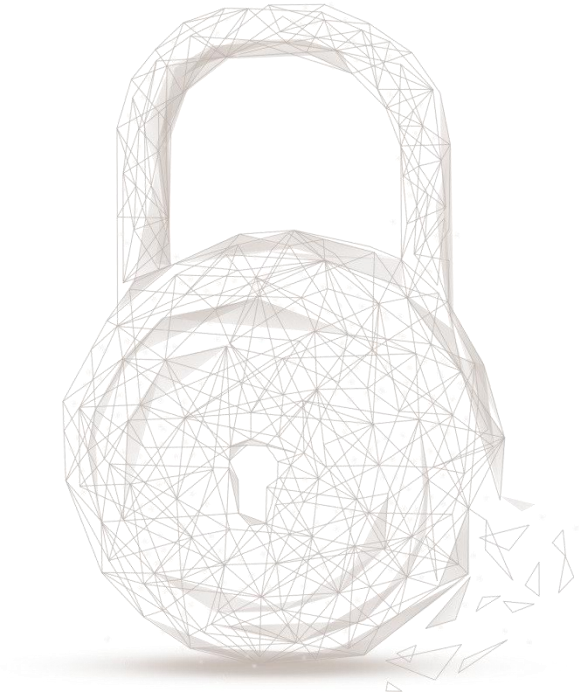
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202102231717

Smart Contract Name：

EHash Token (EHASH)

Smart Contract Address：

0x2942E3B38E33123965bfbc21E802bE943a76bbC6

Smart Contract Address Link：

https://etherscan.io/address/0x2942e3b38e33123965bfbc21e802be943a76bbc6#code

Start Date：2021.02.22

Completion Date：2021.02.23

Overall Result：Pass (Distinction)

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

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	ERC20 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

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 EHASH, including Coding Standards, Security, and Business Logic. **EHASH 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	EHash Token
Token symbol	EHASH
decimals	18
totalSupply	The initial supply is 20 million, mint without cap, burnable.
Token type	ERC20

Table 1 – Basic Token Information

2. Token Vesting Information

N/A

3. Other Feature Description

Mint Revenue:

With regard to mining incentives for contract, 20 percent of mining incentives generated each time belong to managerAddress, users who receive the remaining 80 percent based on EHASH positions. Among them, the reward of knownAddresses is automatically distributed, and the rest of the users need to receive it manually. Users can only receive the reward until the previous round, but not the current round.

Audited Source Code with Comments

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

pragma solidity ^0.6.12; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.


/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
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.
     *
     * IMPORTANT: 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); // Beosin (Chengdu LianAn) // Declare the event 'Transfer'.
    /**
     * @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); // Beosin (Chengdu LianAn) // Declare the event 'Approval'.
}

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

/**
 * @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.
 */
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) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0._
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        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) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message 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.
     *
     * _Available since v2.4.0._
```

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

        return c;
    }

    /**
     * @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) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message 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.
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

/**
 * @dev Collection of functions related to the address type
 */
library Address {
```

```
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies on 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;
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");
```

```
        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
     * `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
```

```
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: value }(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data) internal view returns (bytes memory) {
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data, string memory errorMessage) internal view returns (bytes memory) {
        require(isContract(target), "Address: static call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.staticcall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }
```

```
    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.3._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory) {
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.3._
     */
    function functionDelegateCall(address target, bytes memory data, string memory errorMessage) internal
returns (bytes memory) {
        require(isContract(target), "Address: delegate call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.delegatecall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function _verifyCallResult(bool success, bytes memory returndata, string memory errorMessage) private pure
returns(bytes memory) {
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}
// Beosin (Chengdu LianAn) // Unused library, it is recommended to delete.
library SafeERC20 {
```

```
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    /**
     * @dev Deprecated. This function has issues similar to the ones found in
     * {IERC20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased
allowance below zero");
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing the requirement
     * on the return value: the return value is optional (but if data is returned, it must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism,
```

```
since
        // we're implementing it ourselves. We use {Address.functionCall} to perform this call, which verifies
that
        // the target address contains contract code and also asserts for success in the low-level call.

        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed");
        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}
// Beosin (Chengdu LianAn) // Unused library, it is recommended to delete.
library Strings {
    /**
     * @dev Converts a `uint256` to its ASCII `string` representation.
     */
    function toString(uint256 value) internal pure returns (string memory) {
        // Inspired by OraclizeAPI's implementation - MIT licence
        // https://github.com/oraclize/ethereum-
api/blob/b42146b063c7d6ee1358846c198246239e9360e8/oraclizeAPI_0.4.25.sol

        if (value == 0) {
            return "0";
        }
        uint256 temp = value;
        uint256 digits;
        while (temp != 0) {
            digits++;
            temp /= 10;
        }
        bytes memory buffer = new bytes(digits);
        uint256 index = digits - 1;
        temp = value;
        while (temp != 0) {
            buffer[index--] = byte(uint8(48 + temp % 10));
            temp /= 10;
        }
        return string(buffer);
    }
}
// Beosin (Chengdu LianAn) // Unused contract, it is recommended to delete.
abstract contract ReentrancyGuard {
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;

    uint256 private _status;
```

```
    constructor () public {
        _status = _NOT_ENTERED;
    }

    modifier nonReentrant() {
        // On the first call to nonReentrant, _notEntered will be true
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");

        // Any calls to nonReentrant after this point will fail
        _status = _ENTERED;

        _;

        // By storing the original value once again, a refund is triggered (see
        // https://eips.ethereum.org/EIPS/eip-2200)
        _status = _NOT_ENTERED;
    }
}
// SPDX-License-Identifier: MIT
//

pragma solidity 0.6.12;


/**
 * @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.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner); // Beosin (Chengdu LianAn) // Declare the event 'OwnershipTransferred'.
    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
```

```
    }

    /**
     * @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(_owner == _msgSender(), "Ownable: caller is not the 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.
     */
    // Beosin (Chengdu LianAn) // The renounceOwnership function, owner sets 0 address as the owner of this contract.
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0)); // Beosin (Chengdu LianAn) // Trigger the event 'OwnershipTransferred'.
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    // Beosin (Chengdu LianAn) // Function to set 'newOwner' as owner.
    function transferOwnership(address newOwner) public onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address"); // Beosin (Chengdu LianAn) // Require 'newOwner' not to be 0 address.
        emit OwnershipTransferred(_owner, newOwner); // Beosin (Chengdu LianAn) // Trigger the event 'OwnershipTransferred'.
        _owner = newOwner;
    }
}

/**
 * @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 {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * 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 Context, 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 corresponding address.
    mapping (address => mapping (address => uint256)) private _allowances; // Beosin (Chengdu LianAn) // Declare the mapping variable '_allowances' for storing the allowance between two addresses.

    string public name; // Beosin (Chengdu LianAn) // Declare the variable 'name' for storing the token name.
    string public symbol; // Beosin (Chengdu LianAn) // Declare the variable 'symbol' for storing the token symbol.
    uint8 public decimals; // Beosin (Chengdu LianAn) // Declare the variable 'decimals' for storing the token decimals.
    uint256 private _totalSupply; // Beosin (Chengdu LianAn) // Declare the variable '_totalSupply' for storing the total token supply.

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

    /**
```

```
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view override 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, msg.sender transfers the amount of tokens to a specified address.
    function transfer(address recipient, uint256 amount) public override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view override returns (uint256) {
        return _allowances[owner][spender]; // Beosin (Chengdu LianAn) // Return the approval value 'owner' allowed to 'spender'.
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
  // Beosin (Chengdu LianAn) // Using function 'increaseAllowance' and 'decreaseAllowance' to alter allowance is recommended.
    function approve(address spender, uint256 amount) public override returns (bool) {
        require((allowance(_msgSender(), spender) == 0) || (amount == 0), "ERC20: change allowance use increaseAllowance or decreaseAllowance instead"); // Beosin (Chengdu LianAn) // Using this function to alter the allowance requires that the allowance be reset to 0 first or the allowance between msg.sender and spender is 0.
        _approve(_msgSender(), spender, amount); // Beosin (Chengdu LianAn) // Call internal function '_approve' to alter the allowance between 'msg.sender' and 'spender'.
        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 `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    // Beosin (Chengdu LianAn) // The 'transferFrom' function, msg.sender is a delegate of 'sender' to transfer the amount of tokens to a specified address.
    function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) {
        _transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Call internal function '_transfer' to transfer tokens.
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance")); // Beosin (Chengdu LianAn) // Call internal function '_approve' to alter the allowance between 'sender' and 'msg.sender'.
        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.
     */
    // Beosin (Chengdu LianAn) // The 'increaseAllowance' function, msg.sender increases the allowance which 'msg.sender' allowed to 'spender', altering value is 'addedValue'.
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue)); // Beosin (Chengdu LianAn) // Increase the approval value 'msg.sender' allowed to 'spender'.
        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`.
     */
    // Beosin (Chengdu LianAn) // The 'decreaseAllowance' function, msg.sender decreases the allowance which 'msg.sender' allowed to 'spender', altering value is 'subtractedValue'.
    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero")); // Beosin (Chengdu LianAn) // Decrease the approval value 'msg.sender' allowed to 'spender'.
        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`.
     */
    // Beosin (Chengdu LianAn) // Internal function, implement the 'transfer' operation.
    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'sender'.
        require(recipient != address(0), "ERC20: transfer to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'recipient'.

        _beforeTokenTransfer(sender, recipient, amount);

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance"); // Beosin (Chengdu LianAn) // Alter the balance of 'sender'.
        _balances[recipient] = _balances[recipient].add(amount); // Beosin (Chengdu LianAn) // Alter the 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.
     */
    // Beosin (Chengdu LianAn) // Internal function, implement the 'mint' operation.
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: mint to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.

        _beforeTokenTransfer(address(0), account, amount);

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

    /**
     * @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.
     */
    // Beosin (Chengdu LianAn) // Internal function, implement the 'burn' operation.
    function _burn(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: burn from the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance"); // Beosin (Chengdu LianAn) // Alter the balance of 'account'.
        _totalSupply = _totalSupply.sub(amount); // Beosin (Chengdu LianAn) // Update total Supply of token.
        emit Transfer(account, address(0), amount); // Beosin (Chengdu LianAn) // Trigger the event
```

```
'Transfer'.
    }

    /**
     * @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.
     */
    // Beosin (Chengdu LianAn) // Internal function, implement the 'approve' operation.
    function _approve(address owner, address spender, uint256 amount) internal {
        require(owner != address(0), "ERC20: approve from the zero address"); // Beosin (Chengdu LianAn) //
The non-zero address check for 'owner'.
        require(spender != address(0), "ERC20: approve to the zero address"); // Beosin (Chengdu LianAn) //
The non-zero address check for 'spender'.

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

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be to transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
}

/**
 * @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 Context {
    /**
     * @dev Emitted when the pause is triggered by `account`.
     */
    event Paused(address account); // Beosin (Chengdu LianAn) // Declare the event 'Paused'.

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

    bool private _paused;

    /**
     * @dev Initializes the contract in unpaused state.
     */
    // Beosin (Chengdu LianAn) // Constructor, initialize '_paused' 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.
     *
     * Requirements:
     *
     * - The contract must not be paused.
     */
    modifier whenNotPaused() {
        require(!_paused, "Pausable: paused");
        _;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is paused.
```

```
     *
     * Requirements:
     *
     * - The contract must be paused.
     */
    modifier whenPaused() {
        require(_paused, "Pausable: not paused");
        _;
    }

    /**
     * @dev Triggers stopped state.
     *
     * Requirements:
     *
     * - The contract must not be paused.
     */
    function _pause() internal whenNotPaused {
        _paused = true; // Beosin (Chengdu LianAn) // Alter the variable '_paused' to true.
        emit Paused(_msgSender()); // Beosin (Chengdu LianAn) // Trigger the event 'Paused'.
    }

    /**
     * @dev Returns to normal state.
     *
     * Requirements:
     *
     * - The contract must be paused.
     */
    function _unpause() internal whenPaused {
        _paused = false; // Beosin (Chengdu LianAn) // Alter the variable '_paused' to false.
        emit Unpaused(_msgSender());// Beosin (Chengdu LianAn) // Trigger the event 'Unpaused'.
    }
}

/**
 * @dev EHashBaseToken is the ERC20 extension aspect of EHash
 */
contract EHashBaseToken is ERC20, Pausable, Ownable {
    /**
     * @dev Initialize the contract give all tokens to the deployer
     */
    // Beosin (Chengdu LianAn) // Constructor, initialize the basic information of token.
    constructor(string memory _name, string memory _symbol, uint8 _decimals, uint256 _initialSupply) public {
        name = _name;
        symbol = _symbol;
        decimals = _decimals;
        _mint(_msgSender(), _initialSupply * (10 ** uint256(_decimals)));
```

```
    }

    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See {ERC20-_burn}.
     */
// Beosin (Chengdu LianAn) // Burn function, burn specified amount token of msg.sender.
    function burn(uint256 amount) public {
        _burn(_msgSender(), amount); // Beosin (Chengdu LianAn) // Call internal function '_burn' to burn tokens of msg.sender.
    }

    /** @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.
     */
    // Beosin (Chengdu LianAn) // Mint function, owner can call this function to add the balance of specified address.
    function mint(address account, uint256 amount) public onlyOwner {
        _mint(account,amount); // Beosin (Chengdu LianAn) // Call internal function '_mint' to mint tokens to account address.
    }

    /**
     * @dev Destroys `amount` tokens from `account`, deducting from the caller's
     * allowance.
     *
     * See {ERC20-_burn} and {ERC20-allowance}.
     *
     * Requirements:
     *
     * - the caller must have allowance for ``accounts``'s tokens of at least
     * `amount`.
     */
    // Beosin (Chengdu LianAn) // The 'burnFrom' function, msg.sender is a delegate of 'account' to burn the amount of tokens.
    function burnFrom(address account, uint256 amount) public {
        uint256 decreasedAllowance = allowance(account, _msgSender()).sub(amount, "EHashToken: burn amount exceeds allowance");
        _approve(account, _msgSender(), decreasedAllowance); // Beosin (Chengdu LianAn) // Call internal function '_approve' to decrease allowance.
        _burn(account, amount); // Beosin (Chengdu LianAn) // Call internal function '_burn' to burn
```

```
tokens.
    }

    /**
     * @dev Triggers stopped state.
     * @notice only Owner call
     */
    // Beosin (Chengdu LianAn) // The 'pause' function, owner can pause the contract by calling this function.
    function pause() public onlyOwner {
        _pause();
    }

    /**
     * @dev Returns to normal state.
     * @notice only Owner call
     */
    // Beosin (Chengdu LianAn) // The 'unpause' function, owner can unpause the contract by calling this function.
    function unpause() public onlyOwner {
        _unpause();
    }

    /**
     * @dev Batch transfer amount to recipient
     * @notice that excessive gas consumption causes transaction revert
     */
    // Beosin (Chengdu LianAn) // The 'batchTransfer' function, call this function can transfer token to multiple addresses at the same time, often used in airdrops.
    function batchTransfer(address[] memory recipients, uint256[] memory amounts) public {
        require(recipients.length > 0, "EHashToken: least one recipient address");
        require(recipients.length == amounts.length, "EHashToken: number of recipient addresses does not match the number of tokens");

        for(uint256 i = 0; i < recipients.length; ++i) {
            _transfer(_msgSender(), recipients[i], amounts[i]);
        }
    }
}

/**
 * @dev EHashToken implements EHash mining
 */
contract EHashToken is EHashBaseToken {
    using SafeMath for uint256; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.
    using Address for address payable; // Beosin (Chengdu LianAn) // Use the Address library to check whether the specified address is a contract.
```

```
    // @dev a revenue share multiplier to avert divison downflow.
    uint256 internal constant REVENUE_SHARE_MULTIPLIER = 1e18;

    // @dev update period in secs for revenue distribution.
    uint256 public updatePeriod = 30;

    // @dev for tracking of holders' claimed revenue.
    mapping (address => uint256) internal _revenueClaimed; // Beosin (Chengdu LianAn) // Declare the mapping variable '_revenueClaimed' for storing the claimed revenue of corresponding address.

    /// @dev RoundData always kept for each round.
    struct RoundData {
        uint256 accTokenShare;    // accumulated unit ehash share for each settlement.
        uint256 roundEthers;      // total ethers received in this round.
    }

    /// @dev round index mapping to RoundData.
    mapping (uint => RoundData) private _rounds; // Beosin (Chengdu LianAn) // Declare the mapping variable '_rounds' for storing the RoundData.

    /// @dev mark token holders' highest settled revenue round.
    mapping (address => uint) private _settledRevenueRounds; // Beosin (Chengdu LianAn) // Declare the mapping variable '_settledRevenueRounds' for storing the revenue round of corresponding address.

    /// @dev a monotonic increasing round index, STARTS FROM 1
    uint private _currentRound = 1; // Beosin (Chengdu LianAn) // Declare the variable '_currentRound' for storing the current round information.

    /// @dev expected next update time
    uint private _nextUpdate = block.timestamp + updatePeriod; // Beosin (Chengdu LianAn) // Declare the variable '_nextUpdate' for storing the next time to update.

    /// @dev manager's address
    address payable managerAddress; // Beosin (Chengdu LianAn) // Declare the variable 'managerAddress' for storing the manager's address.

    /// @dev Revenue Claiming log
    event Claim(address indexed account, uint256 amount); // Beosin (Chengdu LianAn) // Declare the event 'Claim'.

    /// @dev Received log
    event Received(address indexed account, uint256 amount); // Beosin (Chengdu LianAn) // Declare the event 'Received'.

    /// @dev known addresss
    address payable [] public knownAddresses; // Beosin (Chengdu LianAn) // Declare the variable 'knownAddresses' for storing the known addresses.
```

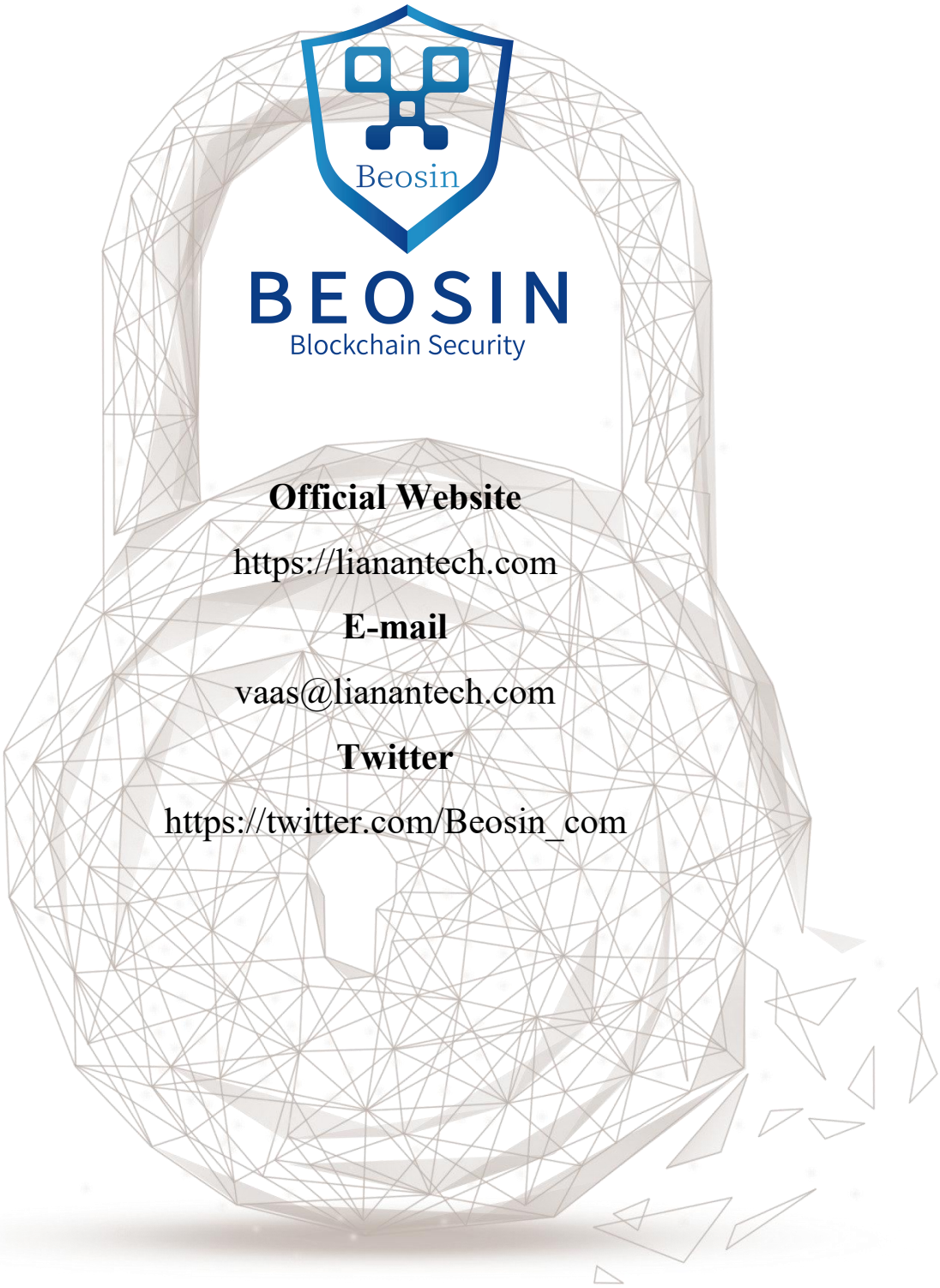
```
    // Beosin (Chengdu LianAn) // Constructor, initialize the basic information of token.
    constructor(string memory _name, string memory _symbol, uint8 _decimals, uint256 _initialSupply)
        EHashBaseToken(_name, _symbol, _decimals, _initialSupply)
        public {
    }

    // @dev tryUpdate function
    // Beosin (Chengdu LianAn) // Modifier 'tryUpdate', require block.timestamp is greater than _nextUpdate.
    modifier tryUpdate() {
        if (block.timestamp > _nextUpdate) {
            update();
        }
        _;
    }

    /**
     * @notice set manager's address
     */
    // Beosin (Chengdu LianAn) // Function 'setManagerAddress', owner can set managerAddress by calling this function.
    function setManagerAddress(address payable account) external onlyOwner {
        require (account != address(0), "0 address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.
        managerAddress = account; // Beosin (Chengdu LianAn) // Set managerAddress to account.
    }

    /**
     * @notice set update period
     */
    // Beosin (Chengdu LianAn) // Function 'setUpdatePeriod', owner can set updatePeriod by calling this function.
    function setUpdatePeriod(uint nsecs) external onlyOwner {
        require (nsecs > 0," period should be positive"); // Beosin (Chengdu LianAn) // Require the nsecs is greater than 0.
        updatePeriod = nsecs; // Beosin (Chengdu LianAn) // Set updatePeriod to nsecs.
    }

    /**
     * @notice set known address
     */
    // Beosin (Chengdu LianAn) // Function 'setKnownAddress', owner can add account to knownAddresses by calling this function.
    function setKnownAddress(address payable account) external onlyOwner {
        removeKnownAddress(account); // Beosin (Chengdu LianAn) // Used to prevent repeated addition of the same address.
        knownAddresses.push(account); // Beosin (Chengdu LianAn) // Add account to known Address.
    }
```

```
    /**
     * @notice remove known address
     */
    // Beosin (Chengdu LianAn) // Function 'removeKnownAddress', owner can remove account from knownAddresses by calling this function.
    function removeKnownAddress(address payable account) public onlyOwner {
        for (uint i=0; i<knownAddresses.length; i++) {
            if (knownAddresses[i] == account) {
                knownAddresses[i] = knownAddresses[knownAddresses.length-1];
                knownAddresses.pop();
                return;
            }
        }
    }

    /**
     * @notice get manager's address
     */
    function getManagerAddress() external view returns(address) {
        return managerAddress;
    }

    /**
     * @notice default ether receiving function
     */
    receive() external payable tryUpdate {
        _rounds[_currentRound].roundEthers += msg.value;
    }

    /**
     * @notice check unclaimed revenue
     */
    // Beosin (Chengdu LianAn) // Function 'checkUnclaimedRevenue', check the unclaimed round revenue of account address.
    function checkUnclaimedRevenue(address account) public view returns(uint256 revenue) {
        uint256 accountTokens = balanceOf(account);
        uint lastSettledRound = _settledRevenueRounds[account];

        uint256 roundRevenue = _rounds[_currentRound-1].accTokenShare.sub(_rounds[lastSettledRound].accTokenShare)
                                                .mul(accountTokens)
                                                .div(REVENUE_SHARE_MULTIPLIER);    // NOTE: div by REVENUE_SHARE_MULTIPLIER

        return roundRevenue;
    }
```

```
    /**
     * @notice get user's life-time gain
     */
    // Beosin (Chengdu LianAn) // Function 'checkTotalRevenue', check the total revenue of account address(Including claimed and unclaimed).
    function checkTotalRevenue(address account) external view returns(uint256 revenue) {
        return checkUnclaimedRevenue(account) + _revenueClaimed[account];
    }

    /**
     * @notice get round N information
     */
    // Beosin (Chengdu LianAn) // Function 'getRoundData', query specified round data.
    function getRoundData(uint round) external view returns(uint256 accTokenShare, uint256 roundEthers) {
        return (_rounds[round].accTokenShare, _rounds[round].roundEthers);
    }

    /**
     * @notice get current round
     */
    // Beosin (Chengdu LianAn) // Function 'getCurrentRound', query current round.
    function getCurrentRound() external view returns(uint round) {
        return _currentRound;
    }

    /**
     * @notice token holders claim revenue
     */
    // Beosin (Chengdu LianAn) // Function 'claim', user can call this function to get the ETH revenue until the last round.
    function claim() external whenNotPaused tryUpdate {
        _claimInternal(msg.sender); // Beosin (Chengdu LianAn) // Call the internal function '_claimInternal' to calculate and send ETH revenue.
    }

    /**
     * @dev internal claim function
     */
    // Beosin (Chengdu LianAn) // Internal function '_claimInternal', implement claim operation.
    function _claimInternal(address payable account) internal {
        uint256 accountTokens = balanceOf(account);
        uint lastSettledRound = _settledRevenueRounds[account];
        uint newSettledRound = _currentRound-1;
        // Beosin (Chengdu LianAn) // Calculate the revenue according to the relevant data.
        uint256 roundRevenue =
_rounds[newSettledRound].accTokenShare.sub(_rounds[lastSettledRound].accTokenShare)
                                              .mul(accountTokens)
                                              .div(REVENUE_SHARE_MULTIPLIER);    // NOTE: div by
```

```
REVENUE_SHARE_MULTIPLIER

            // mark highest settled round revenue claimed.
            _settledRevenueRounds[account] = newSettledRound;
            // Beosin (Chengdu LianAn) // If the award is greater than 0, update the relevant data and send the award.
            if (roundRevenue > 0) {
                // record claimed revenue
                _revenueClaimed[account] += roundRevenue;
                // transfer ETH to account;
                account.sendValue(roundRevenue);
            }
        }

        /**
         * @dev See {ERC20-_beforeTokenTransfer}.
         *
         * Requirements:
         *
         * - the contract must not be paused.
         * - accounts must not trigger the locked `amount` during the locked period.
         */
        // Beosin (Chengdu LianAn) // Internal function '_beforeTokenTransfer', receive the ETH revenue of from and to account of the transaction before transferring the token.
        function _beforeTokenTransfer(address from, address to, uint256 amount) internal override tryUpdate {
            require(!paused(), "EHash: token transfer while paused");

            // handle revenue settlement before token number changes.
            if (from != address(0)) {
                _claimInternal(payable(from));
            }

            if (to != address(0)) {
                _claimInternal(payable(to));
            }

            super._beforeTokenTransfer(from, to, amount);
        }


        /**
         * @dev update function to settle rounds shifting and rounds share.
         */
        // Beosin (Chengdu LianAn) // Internal function 'update', send ETH revenue to manager address and known addresses and then update relevant data.
        // Beosin (Chengdu LianAn) // Note:If the array 'knownAddresses' is too long, this function will consume a lot of gas because of transfer operation.
        function update() internal {
```

```
        // rules:
        // 80% of ethers in this round belongs to all token holders
        //    roundEthers * 80% / totalSupply()
        // and, 20% of ethers belongs to manager
        uint currentRound = _currentRound;
        uint256 roundEthers = _rounds[currentRound].roundEthers;
        uint256 managerRevenue = roundEthers.mul(20).div(100);
        uint256 holdersEthers = roundEthers.sub(managerRevenue);

        // set accmulated holder's share
        _rounds[currentRound].accTokenShare = _rounds[currentRound-1].accTokenShare
                                                .add(
                                                    holdersEthers
                                                    .mul(REVENUE_SHARE_MULTIPLIER) // NOTE: multiplied by REVENUE_SHARE_MULTIPLIER here
                                                    .div(totalSupply())
                                                );

        // next round setting
        _currentRound++;
        _nextUpdate += updatePeriod;

        // send to manager at last
        managerAddress.sendValue(managerRevenue); // Beosin (Chengdu LianAn) // Send revenue to manager address.

        // send to known addreses
        uint numAddresses = knownAddresses.length;
        for (uint i=0; i<numAddresses; i++) {
            _claimInternal(knownAddresses[i]); // Beosin (Chengdu LianAn) // Send revenue to known addresses.
        }
    }

    /**
     * @notice get next update time
     */
    // Beosin (Chengdu LianAn) // Function 'getNextUpdateTime', query the next update time.
    function getNextUpdateTime() external view returns (uint) {
        return _nextUpdate;
    }
}
```



Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com