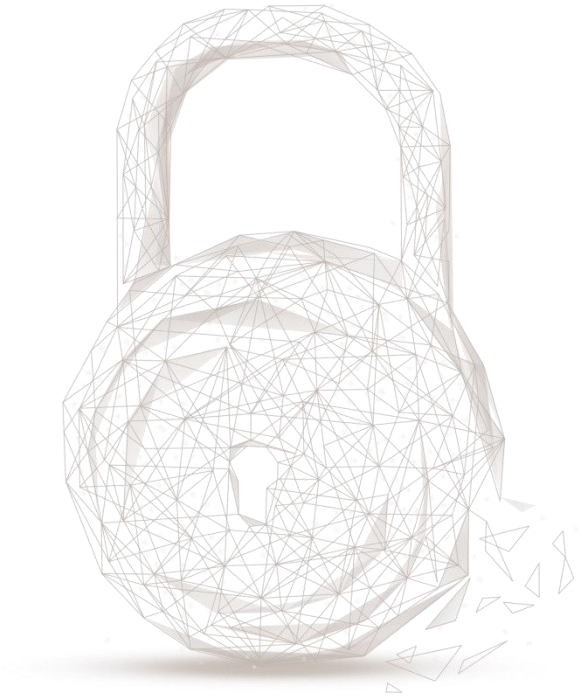
成都链安
BEOSIN

Smart contract security audit report



Audit Number：202008261138

Smart Contract Name：

RAMP DEFI (RAMP)

Smart Contract Address：

0x33D0568941C0C64ff7e0FB4fbA0B11BD37deEd9f

Smart Contract Address Link：

https://etherscan.io/address/0x33d0568941C0C64ff7e0fb4fba0B11BD37deed9f#code

Start Date：2020.08.24

Completion Date：2020.08.26

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	-	Pass
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 RAMP, including Coding Standards, Security, and Business Logic. **RAMP 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	RAMP DEFI
Token symbol	RAMP
decimals	18
totalSupply	1 Billion (The totalSupply is constant)
Token type	ERC20

Table 1 - Basic Token Information

2、Token Vesting Information

This project implements the separate contract VestingPrivateSale or VestingTeam to do the vesting relevant logic. The specified time lock plan of the beneficiary address will be set when deployed the contract VestingPrivateSale or VestingTeam. The tokens to be locked are sent to the corresponding deployed vesting contract address, then transfer the ownership to the beneficiary address.

After the operations above, the time lock is not actually started, after the owner (beneficiary address) set the 'ListingTime', the tokens are locked. They can be released when the current time reaches the corresponding release time.

The detailed lock information is written in source code. Presently, the corresponding contracts are not deployed yet, the following table shows the specified time lock plan:

The beneficiary address	**Lock Amount**	**Release Time**
Specified beneficiary in VestingPrivateSale	36 million	ListingTime + 0 day(not locked)
Specified beneficiary in VestingPrivateSale	36 million	ListingTime + 91 days
Specified beneficiary in VestingPrivateSale	36 million	ListingTime + 182 days
Specified beneficiary in VestingPrivateSale	36 million	ListingTime + 273 days
Specified beneficiary in VestingPrivateSale	36 million	ListingTime + 365 days

Table 2 - Private Sale Vesting Plan (total 180 million)

The beneficiary address	**Lock Amount**	**Release Time**
Specified beneficiary in VestingTeam	26,666,000	ListingTime + 182 days
Specified beneficiary in VestingTeam	26,666,000	ListingTime + 365 days
Specified beneficiary in VestingTeam	26,666,000	ListingTime + 547 days
Specified beneficiary in VestingTeam	26,666,000	ListingTime + 730 days
Specified beneficiary in VestingTeam	26,666,000	ListingTime + 912 days
Specified beneficiary in VestingTeam	26,670,000	ListingTime + 1095 days

Table 3 Team Vesting Plan (total 160 million)

3、Custom Function Audit

- pause/unpause

The token contract implements the functions pause & unpause to control the token transfer. When the contract pause state is true, the token transfer cannot be processed successfully.

- Distributor

The Distributor contract implements the distributeERC20 function to distribute specified ERC20 tokens to specified recipients. The Distributor contract as a delegate of function caller transfers specified tokens to multiple recipients.

Audited Source Code with Comments:

```
/**
 *Submitted for verification at Etherscan.io on 2020-08-21
*/

// File: @openzeppelin/contracts/GSN/Context.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/*
 * @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 {
    // Beosin (Chengdu LianAn) // Internal function '_msgSender' for getting the caller address.
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }
    // Beosin (Chengdu LianAn) // Internal function '_msgData' for returning the call data.
    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;
    }
}

// File: @openzeppelin/contracts/access/Ownable.sol
```

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

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/**
 * @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; // Beosin (Chengdu LianAn) // Declare variable '_owner' for storing the contract 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; // Beosin (Chengdu LianAn) // Initialize the ownership address.
        emit OwnershipTransferred(address(0), msgSender); // Beosin (Chengdu LianAn) // Trigger the event 'OwnershipTransferred'.
    }

    /**
     * @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"); // Beosin (Chengdu LianAn) // Modifier, require that the caller of the modified function must be owner.
        _;
```

```
    }

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

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'newOwner'. Avoid losing ownership.
        emit OwnershipTransferred(_owner, newOwner); // Beosin (Chengdu LianAn) // Trigger the event 'OwnershipTransferred'.
        _owner = newOwner; // Beosin (Chengdu LianAn) // Transfer ownership to 'newOwner'.
    }
}

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    // Beosin (Chengdu LianAn) // Define the function interfaces required by ERC20 Token standard.
    /**
     * @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);
// Beosin (Chengdu LianAn) // Declare the events 'Transfer' and 'Approval'.
/**
 * @dev Emitted when `value` tokens are moved from one account (`from`) to
```

```
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

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

// File: @openzeppelin/contracts/math/SafeMath.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/**
 * @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.
 */
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.
 */
function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    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.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

// File: @openzeppelin/contracts/utils/Address.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.2; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/**
 * @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) {
        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

    /**
     * @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");
        return _functionCallWithValue(target, data, value, errorMessage);
    }

    function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory
errorMessage) private returns (bytes memory) {
        require(isContract(target), "Address: call to non-contract");

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

// File: @openzeppelin/contracts/token/ERC20/ERC20.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.




/**
```

```
 * @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.
    using Address for address; // Beosin (Chengdu LianAn) // Use the Address library for checking the address type. Note: actually is unused.

    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.

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

    string private _name; // Beosin (Chengdu LianAn) // Declare the variable '_name' for storing the token name.
    string private _symbol; // Beosin (Chengdu LianAn) // Declare the variable '_symbol' for storing the token symbol.
    uint8 private _decimals; // Beosin (Chengdu LianAn) // Declare the variable '_decimals' for storing the token decimals.

    /**
     * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
     * a default value of 18.
     *
```

```
 * To select a different value for {decimals}, use {_setupDecimals}.
 *
 * All three of these values are immutable: they can only be set once during
 * construction.
 */
constructor (string memory name, string memory symbol) public {
    _name = name;
    _symbol = symbol;
    _decimals = 18;
}

/**
 * @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. This is the value {ERC20} uses, unless {_setupDecimals} is
 * called.
 *
 * NOTE: 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;
}

/**
 * @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`.
     */
    function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(_msgSender(), recipient, amount); // Beosin (Chengdu LianAn) // Call the internal function '_transfer' to transfer tokens.
        return true;
    }

    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view virtual override 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 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        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`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Call the internal function '_transfer' to transfer tokens.
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance")); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to alter the allowance between two addresses.
        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 virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue)); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to increase the allowance between two addresses.
        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 virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero")); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to decrease the allowance between two addresses.
        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 virtual {
        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'. Avoid losing transferred tokens.

        _beforeTokenTransfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Call the internal function '_beforeTokenTransfer' to check the pause state.

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance"); // 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 virtual {
        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); // Beosin (Chengdu LianAn) // Call the internal function '_beforeTokenTransfer' to check the pause state.

        _totalSupply = _totalSupply.add(amount); // Beosin (Chengdu LianAn) // Update the total token supply.
        _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 Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal virtual {
        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); // Beosin (Chengdu LianAn) // Call the internal function '_beforeTokenTransfer' to check the pause state.

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance"); // Beosin (Chengdu LianAn) // Alter the token balance of 'account'.
        _totalSupply = _totalSupply.sub(amount); // Beosin (Chengdu LianAn) // Update the total token supply.
        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 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 amount) internal virtual {
        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 'amount'.
        emit Approval(owner, spender, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Approval'.
    }

    /**
     * @dev Sets {decimals} to a value other than the default one of 18.
     *
     * WARNING: This function should only be called from the constructor. Most
     * applications that interact with token contracts will not expect
     * {decimals} to ever change, and may work incorrectly if it does.
     */
    function _setupDecimals(uint8 decimals_) internal {
        _decimals = decimals_;
    }

    /**
     * @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 { }
}

// File: @openzeppelin/contracts/utils/Pausable.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.
```

```
/**
 * @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);

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

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

    /**
     * @dev Initializes the contract in unpaused state.
     */
    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 virtual whenNotPaused {
        // Beosin (Chengdu LianAn) // Change the pause status to 'true'.
        _paused = 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 virtual whenPaused {
        // Beosin (Chengdu LianAn) // Change the pause status to 'false'.
        _paused = false;
        emit Unpaused(_msgSender()); // Beosin (Chengdu LianAn) // Trigger the event 'Unpaused'.
    }
}

// File: @openzeppelin/contracts/token/ERC20/ERC20Pausable.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.
```

```
/**
 * @dev ERC20 token with pausable token transfers, minting and burning.
 *
 * Useful for scenarios such as preventing trades until the end of an evaluation
 * period, or having an emergency switch for freezing all token transfers in the
 * event of a large bug.
 */
abstract contract ERC20Pausable is ERC20, Pausable {
    /**
     * @dev See {ERC20-_beforeTokenTransfer}.
     *
     * Requirements:
     *
     * - the contract must not be paused.
     */
    // Beosin (Chengdu LianAn) // Rewrite the internal function '_beforeTokenTransfer', add the pause stake check.
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual override {
        super._beforeTokenTransfer(from, to, amount);

        require(!paused(), "ERC20Pausable: token transfer while paused"); // Beosin (Chengdu LianAn) // Check the pause state, make this function callable only when the contract is not paused.
    }
}

// File: @openzeppelin/contracts/token/ERC20/SafeERC20.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.




/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
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");
        }
    }
}

// File: contracts/Token.sol

//SPDX-License-Identifier: MIT

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.



contract Token is Ownable, ERC20Pausable {
    using SafeERC20 for ERC20Pausable; // Beosin (Chengdu LianAn) // Use the SafeERC20 library for safe external ERC20 contract calling. Note: actually, this library is unused in this contract, it is recommended to delete it.
    // Beosin (Chengdu LianAn) // Constructor, initialize the basic token information and mint tokens to specified 5 addresses.
    constructor(
        address farmingAddress,
        address privateSaleAddress,
        address protocolAddress,
        address publicSaleAddress,
        address teamAddress
    )
        public
        ERC20(
            "RAMP DEFI",       // Name
            "RAMP"             // Symbol
        )
    {
        // Beosin (Chengdu LianAn) // Call the internal function '_mint' to mint tokens to specified 5 addresses. After minting tokens, the total token supply is constant as 1 billion.
        _mint(
            farmingAddress,                 // Farming Reserves multisig
            450000000e18                    // 450,000,000 RAMP
        );
        _mint(
            privateSaleAddress,              // Private Sale multisig
            180000000e18                    // 180,000,000 RAMP
        );
        _mint(
```

```
            protocolAddress,                    // Protocol Reserves multisig
            200000000e18                        // 200,000,000 RAMP
        );
        _mint(
            publicSaleAddress,                   // Public Sale multisig
            10000000e18                         // 10,000,000 RAMP
        );
        _mint(
            teamAddress,                        // Team multisig
            160000000e18                        // 160,000,000 RAMP
        );

        transferOwnership(teamAddress);             // Transfer ownership to the Team multisig
    }

    /**
     * @dev called by the owner to pause, triggers stopped state
     */
    function pause() onlyOwner whenNotPaused public {
        _pause(); // Beosin (Chengdu LianAn) // Call the internal function '_pause' to pause the contract.
    }

    /**
     * @dev called by the owner to unpause, returns to normal state
     */
    function unpause() onlyOwner whenPaused public {
        _unpause(); // Beosin (Chengdu LianAn) // Call the internal function '_unpause' to unpause the contract.
    }
}


// File contracts/Vesting.sol

//SPDX-License-Identifier: MIT

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.



// Vesting provides the main functionality for the vesting approach taken
// The goal is to enable releasing funds from the timelocks after the exchange listing has happened.
// To keep things simple the contract is Ownable, and the owner (a multisig wallet) is able to indicate that
// the exchange listing has happened. All release dates are defined in days relative to the listing date.

// All release schedules are set up during minting, HOWEVER, ERC20 balances will be transferred from the initial
// multisig wallets to the Vesting contracts after minting the ERC20.
// Caveat: The release schedules (timelocks) do need a sufficient balance and will otherwise fail. We decided not
// to write any guards for this situation since it's a 1-time only event and it is easy to remedy (send more RAMP).
```

```
// It will be the responsibility of the RAMP team to fund the Vesting contracts as soon as possible, and with
// the amounts necessary.
// It will also be the responsibility of the RAMP team to call the "setListingTime" function at the appropriate time.

abstract contract Vesting is Ownable {

    // Every timelock has this structure
    struct Timelock {
        address beneficiary;
        uint256 balance;
        uint256 releaseTimeOffset;
    }

    // The timelocks, publicly queryable
    Timelock[] public timelocks;

    // The time of exchange listing, as submitted by the Owner. Starts as 0.
    uint256 public listingTime = 0;

    // The token (RAMP DEFI)
    IERC20 token;

    // Event fired when tokens are released
    event TimelockRelease(address receiver, uint256 amount, uint256 timelock);

    // Vesting is initialized with the token contract
    constructor(address tokenContract) public {
        token = IERC20(tokenContract);
    }

    // Sets up a timelock. Intended to be used during instantiation of an implementing contract
    function setupTimelock(address beneficiary, uint256 amount, uint256 releaseTimeOffset)
        internal
    {
        // Create a variable
        Timelock memory timelock; // Beosin (Chengdu LianAn) // Create the empty time lock plan.

        // Set beneficiary
        timelock.beneficiary = beneficiary; // Beosin (Chengdu LianAn) // Set the beneficiary address.

        // Set balance
        timelock.balance = amount; // Beosin (Chengdu LianAn) // Set the time lock token amount.

        // Set the release time offset. This is a uint256 representing seconds after listingTime
        timelock.releaseTimeOffset = releaseTimeOffset; // Beosin (Chengdu LianAn) // Set the time offset, after
the 'listingTime' is set, the locked tokens can only be released after specified seconds of 'releaseTimeOffset'.
```

```
        // Add the timelock to the array.
        timelocks.push(timelock); // Beosin (Chengdu LianAn) // Store the specified time lock plan.
    }

    // Lets Owner set the listingTime. Can be done only once.
    function setListingTime()
        public
        onlyOwner // Beosin (Chengdu LianAn) // Require that the caller must be the contract owner address.
    {
        // We can run this only once since listingTime will be a timestamp after.
        require(listingTime == 0, "Listingtime was already set");

        // Set the listingtime to the current timestamp.
        listingTime = block.timestamp;
    }

    // Initiates the process to release tokens in a given timelock.
    // Anyone can call this function, but funds will always be released to the beneficiary that was initially set.
    // If the transfer fails for any reason, the transaction will revert.
    // NOTE: It is the RAMP team responsibility to ensure the tokens are indeed owned by this contract.
    function release(uint256 timelockNumber)
        public
    {
        // Check if listingTime is set, otherwise it is not possible to release funds yet.
        require(listingTime > 0, "Listing time was not set yet");

        // Retrieve the requested timelock struct
        Timelock storage timelock = timelocks[timelockNumber];

        // Check if the timelock is ready for release.
        require(listingTime + timelock.releaseTimeOffset <= now, "Timelock can not be released yet.");

        // Get the amount to transfer
        uint256 amount = timelock.balance;

        // Set the timelock balance to 0
        timelock.balance = 0;

        // Transfer the token amount to the beneficiary. If this fails, the transaction will revert.
        require(token.transfer(timelock.beneficiary, amount), "Transfer of amount failed");

        // Emit an event for this.
        emit TimelockRelease(timelock.beneficiary, amount, timelockNumber);

    }

}
```

```
// File contracts/VestingPrivateSale.sol

//SPDX-License-Identifier: MIT

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.



contract VestingPrivateSale is Vesting {
   // Beosin (Chengdu LianAn) // Constructor, initialize the external token contract instance and set specified time lock plan for specified beneficiary address.
   constructor(address tokenContract, address beneficiary) Vesting(tokenContract) public {
      // Beosin (Chengdu LianAn) // Call the internal function 'setupTimelock' to set the different 5 time lock plans for specified beneficiary address.
      // 1 = Private1-5
      setupTimelock(beneficiary, 36000000e18, 0 days);

      // 2 = Private2-5
      setupTimelock(beneficiary, 36000000e18, 91 days);

      // 3 = Private3-5
      setupTimelock(beneficiary, 36000000e18, 182 days);

      // 4 = Private4-5
      setupTimelock(beneficiary, 36000000e18, 273 days);

      // 5 = Private5-5
      setupTimelock(beneficiary, 36000000e18, 365 days);

      // Make the beneficiary (Team multisig) owner of this contract
      transferOwnership(beneficiary); // Beosin (Chengdu LianAn) // Call the function 'transferOwnership' to transfer the ownership to specified beneficiary address.
   }

}


// File contracts/VestingTeam.sol

//SPDX-License-Identifier: MIT

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.



contract VestingTeam is Vesting {
   // Beosin (Chengdu LianAn) // Constructor, initialize the external token contract instance and set specified
```

```
time lock plan for specified beneficiary address.
    constructor(address tokenContract, address beneficiary) Vesting(tokenContract) public {
        // Beosin (Chengdu LianAn) // Call the internal function 'setupTimelock' to set the different 6 time lock plans for specified beneficiary address.
        // 1 = Team1-6
        setupTimelock(beneficiary, 26666000e18, 182 days);

        // 2 = Team2-6
        setupTimelock(beneficiary, 26666000e18, 365 days);

        // 3 = Team3-6
        setupTimelock(beneficiary, 26666000e18, 547 days);

        // 4 = Team4-6
        setupTimelock(beneficiary, 26666000e18, 730 days);

        // 5 = Team5-6
        setupTimelock(beneficiary, 26666000e18, 912 days);

        // 6 = Team6-6
        setupTimelock(beneficiary, 26670000e18, 1095 days);

        // Make the beneficiary (Team multisig) owner of this contract
        transferOwnership(beneficiary); // Beosin (Chengdu LianAn) // Call the function 'transferOwnership' to transfer the ownership to specified beneficiary address.
    }

}


// Sources flattened with buidler v1.4.4 https://buidler.dev

// File @openzeppelin/contracts-ethereum-package/contracts/token/ERC20/IERC20.sol@v3.0.0

pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    // Beosin (Chengdu LianAn) // Define the function interfaces required by ERC20 Token standard.
    /**
     * @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);
// Beosin (Chengdu LianAn) // Declare the events 'Transfer' and 'Approval'.
```

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

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


// File @openzeppelin/upgrades/contracts/Initializable.sol@v2.8.0

pragma solidity >=0.4.24 <0.7.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.


/**
 * @title Initializable
 *
 * @dev Helper contract to support initializer functions. To use it, replace
 * the constructor with a function that has the `initializer` modifier.
 * WARNING: Unlike constructors, initializer functions must be manually
 * invoked. This applies both to deploying an Initializable contract, as well
 * as extending an Initializable contract via inheritance.
 * WARNING: When used with inheritance, manual care must be taken to not invoke
 * a parent initializer twice, or ensure that all initializers are idempotent,
 * because this is not dealt with automatically as with constructors.
 */
contract Initializable {

  /**
   * @dev Indicates that the contract has been initialized.
   */
  bool private initialized; // Beosin (Chengdu LianAn) // Declare the variable 'initialized' for store the initialization finish state.

  /**
   * @dev Indicates that the contract is in the process of being initialized.
   */
  bool private initializing; // Beosin (Chengdu LianAn) // Declare the variable 'initializing' for store the initialization processing state.

  /**
```

```
   * @dev Modifier to use in the initializer function of a contract.
   */
  modifier initializer() {
    require(initializing || isConstructor() || !initialized, "Contract instance has already been initialized"); // Beosin (Chengdu LianAn) // Require that the modified function can be called once.

    bool isTopLevelCall = !initializing;
    if (isTopLevelCall) {
      initializing = true;
      initialized = true;
    }

    _;

    if (isTopLevelCall) {
      initializing = false; // Beosin (Chengdu LianAn) // After executing the modified function logic, change the 'initializing' back to false, making the modified function can only be called once.
    }
  }

  /// @dev Returns true if and only if the function is running in the constructor
  function isConstructor() private view returns (bool) {
    // extcodesize checks the size of the code stored in an address, and
    // address returns the current address. Since the code is still not
    // deployed when running a constructor, any checks on its code size will
    // yield zero, making it an effective way to detect if a contract is
    // under construction or not.
    address self = address(this);
    uint256 cs;
    assembly { cs := extcodesize(self) }
    return cs == 0;
  }

  // Reserved storage space to allow for layout changes in the future.
  uint256[50] private ______gap;
}


// File contracts/Distributor.sol

// contracts/Distributor.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.



contract Distributor is Initializable{
    // Maximum number of recipients. Now set by initializer, maybe changable in a later upgrade.
```

```
    uint8 public maxRecipients;

    // Event for the entire distribution (individual transfers get the ERC20 Transfer event)
    event DistributedERC20(address tokenAddress, uint256 totalAmount);

    // Initializer for the Upgradeable contract (instead of constructor)
    function initialize()
        public
        initializer // Beosin (Chengdu LianAn) // The modifier makes this function can be called once.
    {
        // Initializing maxRecipients to 254
        maxRecipients = 254;
    }

    // Distribute ERC20 token to arrays of recipients/amounts.
    // NOTE: This function needs to have approval already for the total amount to be transferred, and is not checking this.
    function distributeERC20(
        address _tokenAddress,          // Deployment address of the ERC20 contract
        address[] memory _recipients,   // Array of recipient addresses
        uint256[] memory _amounts       // Array of amounts to be sent to each recipient
    )
    public
    {
        // Keep track of the total
        uint256 totalAmountDistributed = 0;

        // Sanity check: array of recipients cannot be longer than the configured maximum
        require(_recipients.length <= maxRecipients);

        // Get a handle on the token
        IERC20 token = IERC20(_tokenAddress);

        // Prepare the iteration variable.
        uint8 i = 0;

        // Loop all the recipients
        for (i; i < _recipients.length; i++) {

            // Transfer ERC20 tokens to the recipient
            require(token.transferFrom(msg.sender, _recipients[i], _amounts[i])); // Beosin (Chengdu LianAn) // Call the 'transferFrom' function of specified token contract to transfer tokens.

            // Increment the total with the amount transferred
            totalAmountDistributed += _amounts[i];

        }
```

```
        // Emit the event that everything is done.
        emit DistributedERC20(_tokenAddress, totalAmountDistributed); // Beosin (Chengdu LianAn) // Trigger the event 'DistributedERC20'.

    }

}
```

成都链安

BEOSIN

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com