kaspersky

Tokeny Smart Contract Code Review and Security Analysis

Kaspersky

18.05.2020

kaspersky

Smart Contract Code Review and Security Analysis Report

Customer: Tokeny Solutions

Date: 06.05.2020

Introduction

Kaspersky was contracted by Tokeny Solutions to conduct a Smart Contract Code Review and Security Analysis. The initial code review was conducted between 29.04.2020 – 06.05.2020.

Tokeny Solutions reacted swiftly and professionally to address issues found, fixed contract was deployed to Ethereum.

Kaspersky performed the second analysis on 15.05.2020 and reevaluated found issues. This report presents the findings of the second security assessment of Customer`s smart contract.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, merely an assessment of its logic and implementation. **The audit does not give any warranties on the security of the code**.

Executive Summary

Kaspersky performed analysis of code functionality, manual audit and automated checks with tools, described in previous section.

Kaspersky confirmed during the code audit, that smart-contract business-description and architecture corresponds to the functionality in the code. The reviewed source codes are well crafted and follow common security practices and compliant with architecture, described in whitepaper.

During the second analysis, we retested identified issues and ensured that Customer fixed the issues or confirmend **that it has no security impact and implemented all recommendations from the report**.

The second Code Review Verdict assigned is green.

More details of contracts are contained in Appendix A.

kaspersky

Scope

On 29.04.2020 contract source codes were obtained

Name	URL
Migrations.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/Migrations.sol
IToken.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/token/IToken.sol
Token.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/token/Token.sol
DefaultCompliance.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/compliance/DefaultCompliance.sol
ICompliance.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/compliance/ICompliance.sol
LimitHolder.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/compliance/LimitHolder.sol
ClaimTopicsRegistry.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/registry/ClaimTopicsRegistry.sol
IClaimTopicsRegistry.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/registry/IClaimTopicsRegistry.sol
IIdentityRegistry.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/registry/IIdentityRegistry.sol
IIdentityRegistryStorage.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/registry/IIdentityRegistryStorage.sol
ITrustedIssuersRegistry.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/registry/ITrustedIssuersRegistry.sol
IdentityRegistry.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/registry/IdentityRegistry.sol
IdentityRegistryStorage.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/registry/IdentityRegistryStorage.sol
TrustedIssuersRegistry.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/registry/TrustedIssuersRegistry.sol
AgentManager.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/roles/AgentManager.sol
AgentRole.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/roles/AgentRole.sol
AgentRoles.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/roles/AgentRoles.sol

kaspersky

Ownable.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/roles/Ownable.sol
OwnerManager.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/roles/OwnerManager.sol
OwnerRoles.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/roles/OwnerRoles.sol
Roles.sol	https://github.com/TokenySolutions/T-REX/blob/beta/contracts/roles/Roles.sol

On 29.04.2020 “T-REX (Token for Regulated EXchanges)” whitepaper was downloaded from specified addresses with their respective MD5 sums

URL	MD5 hash
https://tokeny.com/wp-content/uploads/2020/04/Whitepaper-T-REX-Security-Tokens-V3.pdf	260f50970066870fade69c8010dc36c5

Review Methodology

Throughout the review process, care is taken to ensure that the token contract:

- Implements and adheres to existing ERC-20 Token standard appropriately and effectively
- Documentation and code comments match logic and behavior
- Issues, manages and transfers tokens in a manner that described in in corresponding whitepaper (the smart contract is compliant with the requirement of Customer logic, matching the initial constant values etc.)
- Follows best practices in efficient use of gas, without unnecessary waste
- Uses methods safe from reentrance attacks
- Is not affected by known vulnerabilities

To do so our team of experts review the code line-by-line documenting any issues as they are discovered.

We are scanning this smart contract for commonly known and more specific vulnerabilities. Here are some of the vulnerabilities that are considered (the full list includes them but is not limited to them):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call
- Unchecked math
- Unsafe type inference

kaspersky

- Implicit visibility level

The static analysis portion of our audit is performed using a series of automated tools, purposefully designed to test the security of the contract, such as Remix, Oyente, Solhint.

All the issues are divided into several risk levels:

- Informational - The issue has no impact on the contract's ability to operate (code style violations and info statements, can't affect smart contract execution and can be ignored)
- Low - The issue has minimal impact on the contract's ability to operate (mostly related to outdated, unused etc. code snippets)
- Medium - The issue affects the ability of the contract to operate in a way that doesn't significantly hinder its behavior
- High - High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions
- Critical - The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss

A Code Review Verdict is assigned, which indicates a qualitative score for the Smart Contract source code. Verdict is represented by color mark: red, yellow or green, where

- red indicates the lowest possible source code quality, one or more high/critical issues found
- yellow indicates moderate source code quality, no high/critical issues found and one or more medium issues found
- green indicates the highest possible source code quality, no medium/high/critical issues found

kaspersky

Executive Summary

The reviewed source codes are well crafted and follow common security practices and compliant with architecture, described in whitepaper. We confirmed during the code audit, that smart-contract business-description and architecture corresponds to the functionality in the code.

We performed analysis of code functionality, manual audit and automated checks with tools, described in previous section. All found issues during automated analysis were manually reviewed and applicable vulnerabilities are presented in Issue List section.

During the first analysis we found 3 informational issues, 2 low and 1 medium-level issues. Medium-level issue relies on potential opportunity for attacker to compromise priveledged role to compromise thrustworhiness of platform. Low-level issues are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution. Informational issues code style violations and info statements can't affect smart contract execution and can be ignored.

During the second analysis, we retested identified issues and ensured that Customer fixed the issues or confirmend **that it has no security impact and implemented all recommendations from the report**. The second Code Review Verdict assigned is green.

kaspersky

Appendix A

Reviewed contracts' source code listing can be found below.

Migrations.sol

```
pragma solidity ^0.6.0;

// imports here are just for testing purpose

import "@onchain-id/solidity/contracts/ClaimIssuer.sol";
import "@onchain-id/solidity/contracts/Identity.sol";

contract Migrations {
    address public owner;
    uint public lastCompletedMigration;

    constructor() public {
        owner = msg.sender;
    }

    modifier restricted() {
        if (msg.sender == owner) _;
    }

    function setCompleted(uint completed) public restricted {
        lastCompletedMigration = completed;
    }

    function upgrade(address new_address) public restricted {
        Migrations upgraded = Migrations(new_address);
        upgraded.setCompleted(lastCompletedMigration);
    }
}
```

Token.sol

```
/**
 *     NOTICE
 *
 *     The T-REX software is licensed under a proprietary license or the GPL v.3.
 *     If you choose to receive it under the GPL v.3 license, the following applies:
 *     T-REX is a suite of smart contracts developed by Tokeny to manage and transfer
financial assets on the ethereum blockchain
 *
 *     Copyright (C) 2019, Tokeny sàrl.
 *
 *     This program is free software: you can redistribute it and/or modify
 *     it under the terms of the GNU General Public License as published by
 *     the Free Software Foundation, either version 3 of the License, or
 *     (at your option) any later version.
 *
 *     This program is distributed in the hope that it will be useful,
 *     but WITHOUT ANY WARRANTY; without even the implied warranty of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *     GNU General Public License for more details.
 *
```

```
 *     You should have received a copy of the GNU General Public License
 *     along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */

pragma solidity ^0.6.0;

import "./IToken.sol";
import "@onchain-id/solidity/contracts/IERC734.sol";
import "@onchain-id/solidity/contracts/IERC735.sol";
import "@onchain-id/solidity/contracts/IIdentity.sol";
import "../registry/IClaimTopicsRegistry.sol";
import "../registry/IIdentityRegistry.sol";
import "../compliance/ICompliance.sol";
import "../roles/AgentRole.sol";
import "openzeppelin-solidity/contracts/math/SafeMath.sol";


contract Token is IToken, AgentRole {
    using SafeMath for uint256;

    /// ERC20 basic variables
    mapping(address => uint256) private _balances;
    mapping(address => mapping(address => uint256)) private _allowances;
    uint256 private _totalSupply;

    /// Token information
    string private tokenName;
    string private tokenSymbol;
    uint8 private tokenDecimals;
    address private tokenOnchainID;
    string constant private TOKEN_VERSION = "3.0.0";

    /// Variables of freeze and pause functions
    mapping(address => bool) private frozen;
    mapping(address => uint256) private frozenTokens;

    bool private tokenPaused = false;

    /// Identity Registry contract used by the onchain validator system
    IIdentityRegistry private tokenIdentityRegistry;

    /// Compliance contract linked to the onchain validator system
    ICompliance private tokenCompliance;

   /**
    *  @dev the constructor initiates the token contract
    *  msg.sender is set automatically as the owner of the smart contract
    *  @param _identityRegistry the address of the Identity registry linked to the
token
    *  @param _compliance the address of the compliance contract linked to the token
    *  @param _name the name of the token
    *  @param _symbol the symbol of the token
    *  @param _decimals the decimals of the token
    *  @param _onchainID the address of the onchainID of the token
    *  emits an `UpdatedTokenInformation` event
    *  emits an `IdentityRegistryAdded` event
    *  emits a `ComplianceAdded` event
    */
    constructor(
        address _identityRegistry,
```

```
        address _compliance,
        string memory _name,
        string memory _symbol,
        uint8 _decimals,
        address _onchainID
        )
    public {
        tokenName = _name;
        tokenSymbol = _symbol;
        tokenDecimals = _decimals;
        tokenOnchainID = _onchainID;
        tokenIdentityRegistry = IIdentityRegistry(_identityRegistry);
        emit IdentityRegistryAdded(_identityRegistry);
        tokenCompliance = ICompliance(_compliance);
        emit ComplianceAdded(_compliance);
        emit UpdatedTokenInformation(tokenName, tokenSymbol, tokenDecimals,
TOKEN_VERSION, tokenOnchainID);
    }

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

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

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

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

   /**
    *  @dev See {IERC20-allowance}.
    */
    function allowance(address _owner, address _spender) public override view virtual
returns (uint256) {
        return _allowances[_owner][_spender];
    }

   /**
    *  @dev See {IERC20-approve}.
    */
    function approve(address _spender, uint256 _amount) public override virtual
returns (bool) {
        _approve(msg.sender, _spender, _amount);
        return true;
```

```
    }

    /**
     *  @dev See {ERC20-increaseAllowance}.
     */
    function increaseAllowance(address _spender, uint256 _addedValue) public virtual
returns (bool) {
        _approve(msg.sender, _spender,
_allowances[msg.sender][_spender].add(_addedValue));
        return true;
    }

    /**
     *  @dev See {ERC20-decreaseAllowance}.
     */
    function decreaseAllowance(address _spender, uint256 _subtractedValue) public
virtual returns (bool) {
        _approve(msg.sender, _spender,
_allowances[msg.sender][_spender].sub(_subtractedValue, "ERC20: decreased allowance
below zero"));
        return true;
    }

    /**
     *  @dev See {ERC20-_mint}.
     */
    function _transfer(address _from, address _to, uint256 _amount) internal virtual
{
        require(_from != address(0), "ERC20: transfer from the zero address");
        require(_to != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(_from, _to, _amount);

        _balances[_from] = _balances[_from].sub(_amount, "ERC20: transfer amount
exceeds balance");
        _balances[_to] = _balances[_to].add(_amount);
        emit Transfer(_from, _to, _amount);
    }

    /**
     *  @dev See {ERC20-_mint}.
     */
    function _mint(address _userAddress, uint256 _amount) internal virtual {
        require(_userAddress != address(0), "ERC20: mint to the zero address");

        _beforeTokenTransfer(address(0), _userAddress, _amount);

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

    /**
     *  @dev See {ERC20-_burn}.
     */
    function _burn(address _userAddress, uint256 _amount) internal virtual {
        require(_userAddress != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(_userAddress, address(0), _amount);
```

```
        _balances[_userAddress] = _balances[_userAddress].sub(_amount, "ERC20: burn
amount exceeds balance");
        _totalSupply = _totalSupply.sub(_amount);
        emit Transfer(_userAddress, address(0), _amount);
    }

   /**
    *  @dev See {ERC20-_approve}.
    */
    function _approve(address _owner, address _spender, uint256 _amount) internal
virtual {
        require(_owner != address(0), "ERC20: approve from the zero address");
        require(_spender != address(0), "ERC20: approve to the zero address");

        _allowances[_owner][_spender] = _amount;
        emit Approval(_owner, _spender, _amount);
    }

   /**
    *  @dev See {ERC20-_beforeTokenTransfer}.
    */
    function _beforeTokenTransfer(address _from, address _to, uint256 _amount)
internal virtual { }


   /**
    *  @dev See {IToken-decimals}.
    */
    function decimals() public override view returns (uint8){
        return tokenDecimals;
    }

   /**
    *  @dev See {IToken-name}.
    */
    function name() public override view returns (string memory){
        return tokenName;
    }

   /**
    *  @dev See {IToken-onchainID}.
    */
    function onchainID() public override view returns (address){
        return tokenOnchainID;
    }

   /**
    *  @dev See {IToken-symbol}.
    */
    function symbol() public override view returns (string memory){
        return tokenSymbol;
    }

   /**
    *  @dev See {IToken-version}.
    */
    function version() public override view returns (string memory){
        return TOKEN_VERSION;
    }
```

```
    /**
     *  @dev See {IToken-setName}.
     */
    function setName(string calldata _name) external override onlyOwner {
        tokenName = _name;
        emit UpdatedTokenInformation(tokenName, tokenSymbol, tokenDecimals,
TOKEN_VERSION, tokenOnchainID);
    }

    /**
     *  @dev See {IToken-setSymbol}.
     */
    function setSymbol(string calldata _symbol) external override onlyOwner {
        tokenSymbol = _symbol;
        emit UpdatedTokenInformation(tokenName, tokenSymbol, tokenDecimals,
TOKEN_VERSION, tokenOnchainID);
    }

    /**
     *  @dev See {IToken-setOnchainID}.
     */
    function setOnchainID(address _onchainID) external override onlyOwner {
        tokenOnchainID = _onchainID;
        emit UpdatedTokenInformation(tokenName, tokenSymbol, tokenDecimals,
TOKEN_VERSION, tokenOnchainID);
    }

    /**
     *  @dev See {IToken-paused}.
     */
    function paused() public override view returns (bool) {
        return tokenPaused;
    }

    /**
     *  @dev See {IToken-isFrozen}.
     */
    function isFrozen(address _userAddress) external override view returns (bool) {
        return frozen[_userAddress];
    }

    /**
     *  @dev See {IToken-getFrozenTokens}.
     */
    function getFrozenTokens(address _userAddress) external override view returns
(uint256) {
        return frozenTokens[_userAddress];
    }

    /**
     *  @notice ERC-20 overridden function that include logic to check for trade
validity.
     *  Require that the msg.sender and to addresses are not frozen.
     *  Require that the value should not exceed available balance .
     *  Require that the to address is a verified address
     *  @param _to The address of the receiver
     *  @param _amount The number of tokens to transfer
     *  @return `true` if successful and revert if unsuccessful
     */
```

```
    function transfer(address _to, uint256 _amount) public override whenNotPaused
returns (bool) {
        require(!frozen[_to] && !frozen[msg.sender], "wallet is frozen");
        require(_amount <= balanceOf(msg.sender).sub(frozenTokens[msg.sender]),
"Insufficient Balance");
        if (tokenIdentityRegistry.isVerified(_to) &&
tokenCompliance.canTransfer(msg.sender, _to, _amount)) {
            tokenCompliance.transferred(msg.sender, _to, _amount);
            _transfer(msg.sender, _to, _amount);
            return true;
        }
        revert("Transfer not possible");
    }

   /**
    *  @dev See {IToken-pause}.
    */
    function pause() public override onlyAgent whenNotPaused {
        tokenPaused = true;
        emit Paused(msg.sender);
    }

   /**
    *  @dev See {IToken-unpause}.
    */
    function unpause() public override onlyAgent whenPaused {
        tokenPaused = false;
        emit UnPaused(msg.sender);
    }

   /**
    *  @dev See {IToken-identityRegistry}.
    */
    function identityRegistry() public override view returns (IIdentityRegistry) {
        return tokenIdentityRegistry;
    }

   /**
    *  @dev See {IToken-compliance}.
    */
    function compliance() public override view returns (ICompliance) {
        return tokenCompliance;
    }

   /**
    *  @dev See {IToken-batchTransfer}.
    */
    function batchTransfer(address[] calldata _toList, uint256[] calldata _amounts)
external override {
        for (uint256 i = 0; i < _toList.length; i++) {
            transfer(_toList[i], _amounts[i]);
        }
    }

   /**
    *  @notice ERC-20 overridden function that include logic to check for trade
validity.
    *  Require that the from and to addresses are not frozen.
    *  Require that the value should not exceed available balance .
    *  Require that the to address is a verified address
```

```
     *  @param _from The address of the sender
     *  @param _to The address of the receiver
     *  @param _amount The number of tokens to transfer
     *  @return `true` if successful and revert if unsuccessful
     */
    function transferFrom(address _from, address _to, uint256 _amount) public
override whenNotPaused returns (bool) {
        require(!frozen[_to] && !frozen[_from], "wallet is frozen");
        require(_amount <= balanceOf(_from).sub(frozenTokens[_from]), "Insufficient
Balance");
        if (tokenIdentityRegistry.isVerified(_to) &&
tokenCompliance.canTransfer(_from, _to, _amount)) {
            tokenCompliance.transferred(_from, _to, _amount);
            _transfer(_from, _to, _amount);
            _approve(_from, msg.sender, _allowances[_from][msg.sender].sub(_amount,
"TREX: transfer amount exceeds allowance"));
            return true;
        }

        revert("Transfer not possible");
    }

   /**
     *  @dev See {IToken-forcedTransfer}.
     */
    function forcedTransfer(address _from, address _to, uint256 _amount) public
override onlyAgent returns (bool) {
        uint256 freeBalance = balanceOf(_from).sub(frozenTokens[_from]);
        if (_amount > freeBalance) {
            uint256 tokensToUnfreeze = _amount.sub(freeBalance);
            frozenTokens[_from] = frozenTokens[_from].sub(tokensToUnfreeze);
            emit TokensUnfrozen(_from, tokensToUnfreeze);
        }
        if (tokenIdentityRegistry.isVerified(_to)) {
            tokenCompliance.transferred(_from, _to, _amount);
            _transfer(_from, _to, _amount);
            return true;
        }
        revert("Transfer not possible");
    }

   /**
     *  @dev See {IToken-batchForcedTransfer}.
     */
    function batchForcedTransfer(address[] calldata _fromList, address[] calldata
_toList, uint256[] calldata _amounts) external override {
        for (uint256 i = 0; i < _fromList.length; i++) {
            forcedTransfer(_fromList[i], _toList[i], _amounts[i]);
        }
    }

   /**
     *  @dev See {IToken-mint}.
     */
    function mint(address _to, uint256 _amount) public override onlyAgent {
        require(tokenIdentityRegistry.isVerified(_to), "Identity is not verified.");
        require(tokenCompliance.canTransfer(msg.sender, _to, _amount), "Compliance
not followed");
        _mint(_to, _amount);
        tokenCompliance.created(_to, _amount);
```

```
    }

   /**
    *  @dev See {IToken-batchMint}.
    */
    function batchMint(address[] calldata _toList, uint256[] calldata _amounts)
external override {
        for (uint256 i = 0; i < _toList.length; i++) {
            mint(_toList[i], _amounts[i]);
        }
    }

   /**
    *  @dev See {IToken-burn}.
    */
    function burn(address _userAddress, uint256 _amount) public override onlyAgent {
        uint256 freeBalance = balanceOf(_userAddress) - frozenTokens[_userAddress];
        if (_amount > freeBalance) {
            uint256 tokensToUnfreeze = _amount.sub(freeBalance);
            frozenTokens[_userAddress] =
frozenTokens[_userAddress].sub(tokensToUnfreeze);
            emit TokensUnfrozen(_userAddress, tokensToUnfreeze);
        }
        _burn(_userAddress, _amount);
        tokenCompliance.destroyed(_userAddress, _amount);
    }

   /**
    *  @dev See {IToken-batchBurn}.
    */
    function batchBurn(address[] calldata _userAddresses, uint256[] calldata
_amounts) external override {
        for (uint256 i = 0; i < _userAddresses.length; i++) {
            burn(_userAddresses[i], _amounts[i]);
        }
    }

   /**
    *  @dev See {IToken-setAddressFrozen}.
    */
    function setAddressFrozen(address _userAddress, bool _freeze) public override
onlyAgent {
        frozen[_userAddress] = _freeze;

        emit AddressFrozen(_userAddress, _freeze, msg.sender);
    }

   /**
    *  @dev See {IToken-batchSetAddressFrozen}.
    */
    function batchSetAddressFrozen(address[] calldata _userAddresses, bool[] calldata
_freeze) external override {
        for (uint256 i = 0; i < _userAddresses.length; i++) {
            setAddressFrozen(_userAddresses[i], _freeze[i]);
        }
    }

   /**
    *  @dev See {IToken-freezePartialTokens}.
    */
```

```
    function freezePartialTokens(address _userAddress, uint256 _amount) public
override onlyAgent {
        uint256 balance = balanceOf(_userAddress);
        require(balance >= frozenTokens[_userAddress] + _amount, "Amount exceeds
available balance");
        frozenTokens[_userAddress] = frozenTokens[_userAddress].add(_amount);
        emit TokensFrozen(_userAddress, _amount);
    }

   /**
    *  @dev See {IToken-batchFreezePartialTokens}.
    */
    function batchFreezePartialTokens(address[] calldata _userAddresses, uint256[]
calldata _amounts) external override {
        for (uint256 i = 0; i < _userAddresses.length; i++) {
            freezePartialTokens(_userAddresses[i], _amounts[i]);
        }
    }

   /**
    *  @dev See {IToken-unfreezePartialTokens}.
    */
    function unfreezePartialTokens(address _userAddress, uint256 _amount) public
override onlyAgent {
        require(frozenTokens[_userAddress] >= _amount, "Amount should be less than or
equal to frozen tokens");
        frozenTokens[_userAddress] = frozenTokens[_userAddress].sub(_amount);
        emit TokensUnfrozen(_userAddress, _amount);
    }

   /**
    *  @dev See {IToken-batchUnfreezePartialTokens}.
    */
    function batchUnfreezePartialTokens(address[] calldata _userAddresses, uint256[]
calldata _amounts) external override {
        for (uint256 i = 0; i < _userAddresses.length; i++) {
            unfreezePartialTokens(_userAddresses[i], _amounts[i]);
        }
    }

   /**
    *  @dev See {IToken-setIdentityRegistry}.
    */
    function setIdentityRegistry(address _identityRegistry) public override onlyOwner
{
        tokenIdentityRegistry = IIdentityRegistry(_identityRegistry);
        emit IdentityRegistryAdded(_identityRegistry);
    }

   /**
    *  @dev See {IToken-setCompliance}.
    */
    function setCompliance(address _compliance) public override onlyOwner {
        tokenCompliance = ICompliance(_compliance);
        emit ComplianceAdded(_compliance);
    }

   /**
    *  @dev See {IToken-recoveryAddress}.
    */
```

```
    function recoveryAddress(address _lostWallet, address _newWallet, address
_investorOnchainID) public override onlyAgent returns (bool){
        require(balanceOf(_lostWallet) != 0, "no tokens to recover");
        IIdentity _onchainID = IIdentity(_investorOnchainID);
        bytes32 _key = keccak256(abi.encode(_newWallet));
        if (_onchainID.keyHasPurpose(_key, 1)) {
            uint investorTokens = balanceOf(_lostWallet);
            uint _frozenTokens = frozenTokens[_lostWallet];
            tokenIdentityRegistry.registerIdentity(_newWallet, _onchainID,
tokenIdentityRegistry.investorCountry(_lostWallet));
            tokenIdentityRegistry.deleteIdentity(_lostWallet);
            forcedTransfer(_lostWallet, _newWallet, investorTokens);
            if (_frozenTokens > 0) {
                freezePartialTokens(_newWallet, _frozenTokens);
            }
            if (frozen[_lostWallet] == true) {
                setAddressFrozen(_newWallet, true);
            }
            emit RecoverySuccess(_lostWallet, _newWallet, _investorOnchainID);
            return true;
        }
        revert("Recovery not possible");
    }

   /**
    *  @dev See {IToken-transferOwnershipOnTokenContract}.
    */
    function transferOwnershipOnTokenContract(address _newOwner) public onlyOwner
override {
        transferOwnership(_newOwner);
    }

   /**
    *  @dev See {IToken-addAgentOnTokenContract}.
    */
    function addAgentOnTokenContract(address _agent) external override {
        addAgent(_agent);
    }

   /**
    *  @dev See {IToken-removeAgentOnTokenContract}.
    */
    function removeAgentOnTokenContract(address _agent) external override {
        removeAgent(_agent);
    }
}
```

kaspersky

IToken.sol

```
/**
 *     NOTICE
 *
 *     The T-REX software is licensed under a proprietary license or the GPL v.3.
 *     If you choose to receive it under the GPL v.3 license, the following applies:
 *     T-REX is a suite of smart contracts developed by Tokeny to manage and transfer
financial assets on the ethereum blockchain
 *
 *     Copyright (C) 2019, Tokeny sàrl.
 *
 *     This program is free software: you can redistribute it and/or modify
 *     it under the terms of the GNU General Public License as published by
 *     the Free Software Foundation, either version 3 of the License, or
 *     (at your option) any later version.
 *
 *     This program is distributed in the hope that it will be useful,
 *     but WITHOUT ANY WARRANTY; without even the implied warranty of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *     GNU General Public License for more details.
 *
 *     You should have received a copy of the GNU General Public License
 *     along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */

pragma solidity ^0.6.0;

import "openzeppelin-solidity/contracts/token/ERC20/IERC20.sol";
import "../registry/IIdentityRegistry.sol";
import "../compliance/ICompliance.sol";

///interface
interface IToken is IERC20 {

   /**
    *  this event is emitted when the token information is updated.
    *  the event is emitted by the token constructor and by the setTokenInformation
function
    *  `_newName` is the name of the token
    *  `_newSymbol` is the symbol of the token
    *  `_newDecimals` is the decimals of the token
    *  `_newVersion` is the version of the token, current version is 3.0
    *  `_newOnchainID` is the address of the onchainID of the token
    */
    event UpdatedTokenInformation(string _newName, string _newSymbol, uint8
_newDecimals, string _newVersion, address _newOnchainID);

   /**
    *  this event is emitted when the IdentityRegistry has been set for the token
    *  the event is emitted by the token constructor and by the setIdentityRegistry
function
    *  `_identityRegistry` is the address of the Identity Registry of the token
    */
    event IdentityRegistryAdded(address indexed _identityRegistry);

   /**
    *  this event is emitted when the Compliance has been set for the token
```

```
    *  the event is emitted by the token constructor and by the setCompliance
function
    *  `_compliance` is the address of the Compliance contract of the token
    */
    event ComplianceAdded(address indexed _compliance);

   /**
    *  this event is emitted when an investor successfully recovers his tokens
    *  the event is emitted by the recoveryAddress function
    *  `_lostWallet` is the address of the wallet that the investor lost access to
    *  `_newWallet` is the address of the wallet that the investor provided for the
recovery
    *  `_investorOnchainID` is the address of the onchainID of the investor who asked
for a recovery
    */
    event RecoverySuccess(address _lostWallet, address _newWallet, address
_investorOnchainID);

   /**
    *  this event is emitted when the wallet of an investor is frozen or unfrozen
    *  the event is emitted by setAddressFrozen and batchSetAddressFrozen functions
    *  `_userAddress` is the wallet of the investor that is concerned by the freezing
status
    *  `_isFrozen` is the freezing status of the wallet
    *  if `_isFrozen` equals `true` the wallet is frozen after emission of the event
    *  if `_isFrozen` equals `false` the wallet is unfrozen after emission of the
event
    *  `_owner` is the address of the agent who called the function to freeze the
wallet
    */
    event AddressFrozen(address indexed _userAddress, bool indexed _isFrozen, address
indexed _owner);

   /**
    *  this event is emitted when a certain amount of tokens is frozen on a wallet
    *  the event is emitted by freezePartialTokens and batchFreezePartialTokens
functions
    *  `_userAddress` is the wallet of the investor that is concerned by the freezing
status
    *  `_amount` is the amount of tokens that are frozen
    */
    event TokensFrozen(address indexed _userAddress, uint256 _amount);

   /**
    *  this event is emitted when a certain amount of tokens is unfrozen on a wallet
    *  the event is emitted by unfreezePartialTokens and batchUnfreezePartialTokens
functions
    *  `_userAddress` is the wallet of the investor that is concerned by the freezing
status
    *  `_amount` is the amount of tokens that are unfrozen
    */
    event TokensUnfrozen(address indexed _userAddress, uint256 _amount);

   /**
    *  this event is emitted when the token is paused
    *  the event is emitted by the pause function
    *  `_userAddress` is the address of the wallet that called the pause function
    */
    event Paused(address _userAddress);
```

```
/**
 *  this event is emitted when the token is unpaused
 *  the event is emitted by the unpause function
 *  `_userAddress` is the address of the wallet that called the unpause function
 */
 event UnPaused(address _userAddress);

/**
 * @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 / 1 ** 2`).
 *
 * Tokens usually opt for a value of 18, imitating the relationship between
 * Ether and Wei.
 *
 * NOTE: This information is only used for _display_ purposes: it in
 * no way affects any of the arithmetic of the contract, including
 * balanceOf() and transfer().
 */
 function decimals() external view returns (uint8);

/**
 * @dev Returns the name of the token.
 */
 function name() external view returns (string memory);

/**
 * @dev Returns the address of the onchainID of the token.
 * the onchainID of the token gives all the information available
 * about the token and is managed by the token issuer or his agent.
 */
 function onchainID() external view returns (address);

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

/**
 * @dev Returns the TREX version of the token.
 * current version is 3.0.0
 */
 function version() external view returns (string memory);

/**
 *  @dev Returns the Identity Registry linked to the token
 */
 function identityRegistry() external view returns (IIdentityRegistry);

/**
 *  @dev Returns the Compliance contract linked to the token
 */
 function compliance() external view returns (ICompliance);

/**
 * @dev Returns true if the contract is paused, and false otherwise.
 */
 function paused() external view returns (bool);
```

```
    /**
     *  @dev Returns the freezing status of a wallet
     *  if isFrozen returns `true` the wallet is frozen
     *  if isFrozen returns `false` the wallet is not frozen
     *  isFrozen returning `true` doesn't mean that the balance is free, tokens could
be blocked by
     *  a partial freeze or the whole token could be blocked by pause
     *  @param _userAddress the address of the wallet on which isFrozen is called
     */
    function isFrozen(address _userAddress) external view returns (bool);

    /**
     *  @dev Returns the amount of tokens that are partially frozen on a wallet
     *  the amount of frozen tokens is always <= to the total balance of the wallet
     *  @param _userAddress the address of the wallet on which getFrozenTokens is
called
     */
    function getFrozenTokens(address _userAddress) external view returns (uint256);

    /**
     *  @dev sets the token name
     *  @param _name the name of token to set
     *  Only the owner of the token smart contract can call this function
     *  emits a `UpdatedTokenInformation` event
     */
    function setName(string calldata _name) external;

    /**
     *  @dev sets the token symbol
     *  @param _symbol the token symbol to set
     *  Only the owner of the token smart contract can call this function
     *  emits a `UpdatedTokenInformation` event
     */
    function setSymbol(string calldata _symbol) external;

    /**
     *  @dev sets the onchain ID of the token
     *  @param _onchainID the address of the onchain ID to set
     *  Only the owner of the token smart contract can call this function
     *  emits a `UpdatedTokenInformation` event
     */
    function setOnchainID(address _onchainID) external;

    /**
     *  @dev pauses the token contract, when contract is paused investors cannot
transfer tokens anymore
     *  This function can only be called by a wallet set as agent of the token
     *  emits a `Paused` event
     */
    function pause() external;

    /**
     *  @dev unpauses the token contract, when contract is unpaused investors can
transfer tokens
     *  if their wallet is not blocked & if the amount to transfer is <= to the amount
of free tokens
     *  This function can only be called by a wallet set as agent of the token
     *  emits an `Unpaused` event
     */
    function unpause() external;
```

```
    /**
     *  @dev sets an address frozen status for this token.
     *  @param _userAddress The address for which to update frozen status
     *  @param _freeze Frozen status of the address
     *  This function can only be called by a wallet set as agent of the token
     *  emits an `AddressFrozen` event
     */
     function setAddressFrozen(address _userAddress, bool _freeze) external;

    /**
     *  @dev freezes token amount specified for given address.
     *  @param _userAddress The address for which to update frozen tokens
     *  @param _amount Amount of Tokens to be frozen
     *  This function can only be called by a wallet set as agent of the token
     *  emits a `TokensFrozen` event
     */
     function freezePartialTokens(address _userAddress, uint256 _amount) external;

    /**
     *  @dev unfreezes token amount specified for given address
     *  @param _userAddress The address for which to update frozen tokens
     *  @param _amount Amount of Tokens to be unfrozen
     *  This function can only be called by a wallet set as agent of the token
     *  emits a `TokensUnfrozen` event
     */
     function unfreezePartialTokens(address _userAddress, uint256 _amount) external;

    /**
     *  @dev sets the Identity Registry for the token
     *  @param _identityRegistry the address of the Identity Registry to set
     *  Only the owner of the token smart contract can call this function
     *  emits an `IdentityRegistryAdded` event
     */
     function setIdentityRegistry(address _identityRegistry) external;

    /**
     *  @dev sets the compliance contract of the token
     *  @param _compliance the address of the compliance contract to set
     *  Only the owner of the token smart contract can call this function
     *  emits a `ComplianceAdded` event
     */
     function setCompliance(address _compliance) external;

    /**
     *  @dev force a transfer of tokens between 2 whitelisted wallets
     *  In case the `from` address has not enough free tokens (unfrozen tokens)
     *  but has a total balance higher or equal to the `amount`
     *  the amount of frozen tokens is reduced in order to have enough free tokens
     *  to proceed the transfer, in such a case, the remaining balance on the `from`
     *  account is 100% composed of frozen tokens post-transfer.
     *  Require that the `to` address is a verified address,
     *  @param _from The address of the sender
     *  @param _to The address of the receiver
     *  @param _amount The number of tokens to transfer
     *  @return `true` if successful and revert if unsuccessful
     *  This function can only be called by a wallet set as agent of the token
     *  emits a `TokensUnfrozen` event if `_amount` is higher than the free balance of
`_from`
     *  emits a `Transfer` event
```

```
    */
    function forcedTransfer(address _from, address _to, uint256 _amount) external
returns (bool);

   /**
    *  @dev mint tokens on a wallet
    *  Improved version of default mint method. Tokens can be minted
    *  to an address if only it is a verified address as per the security token.
    *  @param _to Address to mint the tokens to.
    *  @param _amount Amount of tokens to mint.
    *  This function can only be called by a wallet set as agent of the token
    *  emits a `Transfer` event
    */
    function mint(address _to, uint256 _amount) external;

   /**
    *  @dev burn tokens on a wallet
    *  In case the `account` address has not enough free tokens (unfrozen tokens)
    *  but has a total balance higher or equal to the `value` amount
    *  the amount of frozen tokens is reduced in order to have enough free tokens
    *  to proceed the burn, in such a case, the remaining balance on the `account`
    *  is 100% composed of frozen tokens post-transaction.
    *  @param _userAddress Address to burn the tokens from.
    *  @param _amount Amount of tokens to burn.
    *  This function can only be called by a wallet set as agent of the token
    *  emits a `TokensUnfrozen` event if `_amount` is higher than the free balance of
`_userAddress`
    *  emits a `Transfer` event
    */
    function burn(address _userAddress, uint256 _amount) external;

   /**
    *  @dev recovery function used to force transfer tokens from a
    *  lost wallet to a new wallet for an investor.
    *  @param _lostWallet the wallet that the investor lost
    *  @param _newWallet the newly provided wallet on which tokens have to be
transferred
    *  @param _investorOnchainID the onchainID of the investor asking for a recovery
    *  This function can only be called by a wallet set as agent of the token
    *  emits a `TokensUnfrozen` event if there is some frozen tokens on the lost
wallet if the recovery process is successful
    *  emits a `Transfer` event if the recovery process is successful
    *  emits a `RecoverySuccess` event if the recovery process is successful
    *  emits a `RecoveryFails` event if the recovery process fails
    */
    function recoveryAddress(address _lostWallet, address _newWallet, address
_investorOnchainID) external returns (bool);

   /**
    *  @dev function allowing to issue transfers in batch
    *  Require that the msg.sender and `to` addresses are not frozen.
    *  Require that the total value should not exceed available balance.
    *  Require that the `to` addresses are all verified addresses,
    *  IMPORTANT : THIS TRANSACTION COULD EXCEED GAS LIMIT IF `_toList.length` IS TOO
HIGH,
    *  USE WITH CARE OR YOU COULD LOSE TX FEES WITH AN "OUT OF GAS" TRANSACTION
    *  @param _toList The addresses of the receivers
    *  @param _amounts The number of tokens to transfer to the corresponding receiver
    *  emits _toList.length `Transfer` events
    */
```

```
    function batchTransfer(address[] calldata _toList, uint256[] calldata _amounts)
external;

   /**
    *  @dev function allowing to issue forced transfers in batch
    *  Require that `_amounts[i]` should not exceed available balance of
`_fromList[i]`.
    *  Require that the `_toList` addresses are all verified addresses
    *  IMPORTANT : THIS TRANSACTION COULD EXCEED GAS LIMIT IF `_fromList.length` IS
TOO HIGH,
    *  USE WITH CARE OR YOU COULD LOSE TX FEES WITH AN "OUT OF GAS" TRANSACTION
    *  @param _fromList The addresses of the senders
    *  @param _toList The addresses of the receivers
    *  @param _amounts The number of tokens to transfer to the corresponding receiver
    *  This function can only be called by a wallet set as agent of the token
    *  emits `TokensUnfrozen` events if `_amounts[i]` is higher than the free balance
of `_fromList[i]`
    *  emits _fromList.length `Transfer` events
    */
    function batchForcedTransfer(address[] calldata _fromList, address[] calldata
_toList, uint256[] calldata _amounts) external;

   /**
    *  @dev function allowing to mint tokens in batch
    *  Require that the `_toList` addresses are all verified addresses
    *  IMPORTANT : THIS TRANSACTION COULD EXCEED GAS LIMIT IF `_toList.length` IS TOO
HIGH,
    *  USE WITH CARE OR YOU COULD LOSE TX FEES WITH AN "OUT OF GAS" TRANSACTION
    *  @param _toList The addresses of the receivers
    *  @param _amounts The number of tokens to mint to the corresponding receiver
    *  This function can only be called by a wallet set as agent of the token
    *  emits _toList.length `Transfer` events
    */
    function batchMint(address[] calldata _toList, uint256[] calldata _amounts)
external;

   /**
    *  @dev function allowing to burn tokens in batch
    *  Require that the `_userAddresses` addresses are all verified addresses
    *  IMPORTANT : THIS TRANSACTION COULD EXCEED GAS LIMIT IF `_userAddresses.length`
IS TOO HIGH,
    *  USE WITH CARE OR YOU COULD LOSE TX FEES WITH AN "OUT OF GAS" TRANSACTION
    *  @param _userAddresses The addresses of the wallets concerned by the burn
    *  @param _amounts The number of tokens to burn from the corresponding wallets
    *  This function can only be called by a wallet set as agent of the token
    *  emits _userAddresses.length `Transfer` events
    */
    function batchBurn(address[] calldata _userAddresses, uint256[] calldata
_amounts) external;

   /**
    *  @dev function allowing to set frozen addresses in batch
    *  IMPORTANT : THIS TRANSACTION COULD EXCEED GAS LIMIT IF `_userAddresses.length`
IS TOO HIGH,
    *  USE WITH CARE OR YOU COULD LOSE TX FEES WITH AN "OUT OF GAS" TRANSACTION
    *  @param _userAddresses The addresses for which to update frozen status
    *  @param _freeze Frozen status of the corresponding address
    *  This function can only be called by a wallet set as agent of the token
    *  emits _userAddresses.length `AddressFrozen` events
    */
```

```
    function batchSetAddressFrozen(address[] calldata _userAddresses, bool[] calldata
_freeze) external;

   /**
    *  @dev function allowing to freeze tokens partially in batch
    *  IMPORTANT : THIS TRANSACTION COULD EXCEED GAS LIMIT IF `_userAddresses.length`
IS TOO HIGH,
    *  USE WITH CARE OR YOU COULD LOSE TX FEES WITH AN "OUT OF GAS" TRANSACTION
    *  @param _userAddresses The addresses on which tokens need to be frozen
    *  @param _amounts the amount of tokens to freeze on the corresponding address
    *  This function can only be called by a wallet set as agent of the token
    *  emits _userAddresses.length `TokensFrozen` events
    */
    function batchFreezePartialTokens(address[] calldata _userAddresses, uint256[]
calldata _amounts) external;

   /**
    *  @dev function allowing to unfreeze tokens partially in batch
    *  IMPORTANT : THIS TRANSACTION COULD EXCEED GAS LIMIT IF `_userAddresses.length`
IS TOO HIGH,
    *  USE WITH CARE OR YOU COULD LOSE TX FEES WITH AN "OUT OF GAS" TRANSACTION
    *  @param _userAddresses The addresses on which tokens need to be unfrozen
    *  @param _amounts the amount of tokens to unfreeze on the corresponding address
    *  This function can only be called by a wallet set as agent of the token
    *  emits _userAddresses.length `TokensUnfrozen` events
    */
    function batchUnfreezePartialTokens(address[] calldata _userAddresses, uint256[]
calldata _amounts) external;

   /**
    *  @dev transfers the ownership of the token smart contract
    *  @param _newOwner the address of the new token smart contract owner
    *  This function can only be called by the owner of the token
    *  emits an `OwnershipTransferred` event
    */
    function transferOwnershipOnTokenContract(address _newOwner) external;

   /**
    *  @dev adds an agent to the token smart contract
    *  @param _agent the address of the new agent of the token smart contract
    *  This function can only be called by the owner of the token
    *  emits an `AgentAdded` event
    */
    function addAgentOnTokenContract(address _agent) external;

   /**
    *  @dev remove an agent from the token smart contract
    *  @param _agent the address of the agent to remove
    *  This function can only be called by the owner of the token
    *  emits an `AgentRemoved` event
    */
    function removeAgentOnTokenContract(address _agent) external;

}
```

DefaultCompliance.sol

```
/**
 *     NOTICE
```

```
 *
 *     The T-REX software is licensed under a proprietary license or the GPL v.3.
 *     If you choose to receive it under the GPL v.3 license, the following applies:
 *     T-REX is a suite of smart contracts developed by Tokeny to manage and transfer
financial assets on the ethereum blockchain
 *
 *     Copyright (C) 2019, Tokeny sàrl.
 *
 *     This program is free software: you can redistribute it and/or modify
 *     it under the terms of the GNU General Public License as published by
 *     the Free Software Foundation, either version 3 of the License, or
 *     (at your option) any later version.
 *
 *     This program is distributed in the hope that it will be useful,
 *     but WITHOUT ANY WARRANTY; without even the implied warranty of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *     GNU General Public License for more details.
 *
 *     You should have received a copy of the GNU General Public License
 *     along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */

pragma solidity ^0.6.0;

import "./ICompliance.sol";

import "../roles/Ownable.sol";

contract DefaultCompliance is ICompliance, Ownable {

   /**
    *  @dev See {ICompliance-canTransfer}.
    */
    function canTransfer(address _from, address _to, uint256 _value) public override
view returns (bool) {
        return true;
    }

   /**
    *  @dev See {ICompliance-transferred}.
    */
    function transferred(address _from, address _to, uint256 _value) public override
{

    }

   /**
    *  @dev See {ICompliance-created}.
    */
    function created(address _to, uint256 _value) public override {

    }

   /**
    *  @dev See {ICompliance-destroyed}.
    */
    function destroyed(address _from, uint256 _value) public override {

    }
```

```
   /**
    *  @dev See {ICompliance-transferOwnershipOnComplianceContract}.
    */
    function transferOwnershipOnComplianceContract(address newOwner) external
override onlyOwner {
        transferOwnership(newOwner);
    }
}
```

ICompliance.sol

```
/**
 *     NOTICE
 *
 *     The T-REX software is licensed under a proprietary license or the GPL v.3.
 *     If you choose to receive it under the GPL v.3 license, the following applies:
 *     T-REX is a suite of smart contracts developed by Tokeny to manage and transfer
financial assets on the ethereum blockchain
 *
 *     Copyright (C) 2019, Tokeny sàrl.
 *
 *     This program is free software: you can redistribute it and/or modify
 *     it under the terms of the GNU General Public License as published by
 *     the Free Software Foundation, either version 3 of the License, or
 *     (at your option) any later version.
 *
 *     This program is distributed in the hope that it will be useful,
 *     but WITHOUT ANY WARRANTY; without even the implied warranty of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *     GNU General Public License for more details.
 *
 *     You should have received a copy of the GNU General Public License
 *     along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */

pragma solidity ^0.6.0;

interface ICompliance {

   /**
    *  @dev checks that the transfer is compliant.
    *  default compliance always returns true
    *  READ ONLY FUNCTION, this function cannot be used to increment
    *  counters, emit events, ...
    *  @param _from The address of the sender
    *  @param _to The address of the receiver
    *  @param _amount The amount of tokens involved in the transfer
    */
    function canTransfer(address _from, address _to, uint256 _amount) external view
returns (bool);

   /**
    *  @dev function called whenever tokens are transferred
    *  from one wallet to another
    *  this function can update state variables in the compliance contract
    *  these state variables being used by `canTransfer` to decide if a transfer
    *  is compliant or not depending on the values stored in these state variables
and on
```

```
     *  the parameters of the compliance smart contract
     *  @param _from The address of the sender
     *  @param _to The address of the receiver
     *  @param _amount The amount of tokens involved in the transfer
     */
    function transferred(address _from, address _to, uint256 _amount) external;

   /**
     *  @dev function called whenever tokens are created
     *  on a wallet
     *  this function can update state variables in the compliance contract
     *  these state variables being used by `canTransfer` to decide if a transfer
     *  is compliant or not depending on the values stored in these state variables
and on
     *  the parameters of the compliance smart contract
     *  @param _to The address of the receiver
     *  @param _amount The amount of tokens involved in the transfer
     */
    function created(address _to, uint256 _amount) external;

   /**
     *  @dev function called whenever tokens are destroyed
     *  this function can update state variables in the compliance contract
     *  these state variables being used by `canTransfer` to decide if a transfer
     *  is compliant or not depending on the values stored in these state variables
and on
     *  the parameters of the compliance smart contract
     *  @param _from The address of the receiver
     *  @param _amount The amount of tokens involved in the transfer
     */
    function destroyed(address _from, uint256 _amount) external;

   /**
     *  @dev function used to transfer the ownership of the compliance contract
     *  to a new owner, giving him access to the `OnlyOwner` functions implemented on
the contract
     *  @param newOwner The address of the new owner of the compliance contract
     *  This function can only be called by the owner of the compliance contract
     *  emits an `OwnershipTransferred` event
     */
    function transferOwnershipOnComplianceContract(address newOwner) external;
}
```

LimitHolder.sol

```
/**
 *     NOTICE
 *
 *     The T-REX software is licensed under a proprietary license or the GPL v.3.
 *     If you choose to receive it under the GPL v.3 license, the following applies:
 *     T-REX is a suite of smart contracts developed by Tokeny to manage and transfer
financial assets on the ethereum blockchain
 *
 *     Copyright (C) 2019, Tokeny sàrl.
 *
 *     This program is free software: you can redistribute it and/or modify
 *     it under the terms of the GNU General Public License as published by
```

```
 *     the Free Software Foundation, either version 3 of the License, or
 *     (at your option) any later version.
 *
 *     This program is distributed in the hope that it will be useful,
 *     but WITHOUT ANY WARRANTY; without even the implied warranty of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *     GNU General Public License for more details.
 *
 *     You should have received a copy of the GNU General Public License
 *     along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */

pragma solidity ^0.6.0;

import "./ICompliance.sol";
import "../token/IToken.sol";
import "../roles/AgentRole.sol";
import "../registry/IIdentityRegistry.sol";

contract LimitHolder is ICompliance, AgentRole {

    /// the token on which this compliance contract is applied
    IToken public token;

    /// the limit of holders for this token
    uint private holderLimit;

    /// the Identity registry contract linked to `token`
    IIdentityRegistry private identityRegistry;

    /// the index of each shareholder in the array `shareholders`
    mapping(address => uint256) private holderIndices;

    /// the amount of shareholders per country
    mapping(uint16 => uint256) private countryShareHolders;

    /// the addresses of all shareholders
    address[] private shareholders;

   /**
    *  this event is emitted when the holder limit is set.
    *  the event is emitted by the setHolderLimit function and by the constructor
    *  `_holderLimit` is the holder limit for this token
    */
    event HolderLimitSet (uint _holderLimit);

   /**
    *  @dev the constructor initiates the smart contract with the initial state
variables
    *  @param _token the address of the token concerned by the rules of this
compliance contract
    *  @param _holderLimit the holder limit for the token concerned
    *  emits a `HolderLimitSet` event
    */
    constructor (address _token, uint _holderLimit) public {
        token = IToken(_token);
        holderLimit = _holderLimit;
        identityRegistry = token.identityRegistry();
        emit HolderLimitSet(_holderLimit);
    }
```

```
/**
 *  @dev sets the holder limit as required for compliance purpose
 *  @param _holderLimit the holder limit for the token concerned
 *  This function can only be called by the agent of the Compliance contract
 *  emits a `HolderLimitSet` event
 */
function setHolderLimit(uint _holderLimit) public onlyAgent {
    holderLimit = _holderLimit;
    emit HolderLimitSet(_holderLimit);
}

/**
 *  @dev returns the holder limit as set on the contract
 */
function getHolderLimit() public view returns (uint) {
    return holderLimit;
}


/**
 *  @dev returns the amount of token holders
 */
function holderCount() public view returns (uint) {
    return shareholders.length;
}

/**
 *  @dev By counting the number of token holders using `holderCount`
 *  you can retrieve the complete list of token holders, one at a time.
 *  It MUST throw if `index >= holderCount()`.
 *  @param index The zero-based index of the holder.
 *  @return `address` the address of the token holder with the given index.
 */
function holderAt(uint256 index) public view returns (address){
    require(index < shareholders.length, "shareholder doesn't exist");
    return shareholders[index];
}


/**
 *  @dev If the address is not in the `shareholders` array then push it
 *  and update the `holderIndices` mapping.
 *  @param addr The address to add as a shareholder if it's not already.
 */
function updateShareholders(address addr) internal {
    if (holderIndices[addr] == 0) {
        shareholders.push(addr);
        holderIndices[addr] = shareholders.length;
        uint16 country = identityRegistry.investorCountry(addr);
        countryShareHolders[country]++;
    }
}

/**
 *  If the address is in the `shareholders` array and the forthcoming
 *  transfer or transferFrom will reduce their balance to 0, then
 *  we need to remove them from the shareholders array.
 *  @param addr The address to prune if their balance will be reduced to 0.
 *  @dev see https://ethereum.stackexchange.com/a/39311
```

```
     */
    function pruneShareholders(address addr) internal {
        require(holderIndices[addr] != 0, "Shareholder does not exist");
        uint256 balance = token.balanceOf(addr);
        if (balance > 0) {
            return;
        }
        uint256 holderIndex = holderIndices[addr] - 1;
        uint256 lastIndex = shareholders.length - 1;
        address lastHolder = shareholders[lastIndex];
        shareholders[holderIndex] = lastHolder;
        holderIndices[lastHolder] = holderIndices[addr];
        shareholders.pop();
        holderIndices[addr] = 0;
        uint16 country = identityRegistry.investorCountry(addr);
        countryShareHolders[country]--;
    }

   /**
    *  @dev get the amount of shareholders in a country
    *  @param index the index of the country, following ISO 3166-1
    */
    function getShareholderCountByCountry(uint16 index) public view returns (uint) {
        return countryShareHolders[index];
    }


   /**
    *  @dev See {ICompliance-canTransfer}.
    *  @return true if the amount of holders post-transfer is less or
    *  equal to the maximum amount of token holders
    */
    function canTransfer(address _from, address _to, uint256 _value) public override
view returns (bool) {
        if (holderIndices[_to] != 0) {
            return true;
        }
        if (holderCount() < holderLimit) {
            return true;
        }
        return false;
    }

   /**
    *  @dev See {ICompliance-transferred}.
    *  updates the counter of shareholders if necessary
    */
    function transferred(address _from, address _to, uint256 _value) public override
onlyAgent {
        updateShareholders(_to);
        pruneShareholders(_from);
    }

   /**
    *  @dev See {ICompliance-created}.
    *  updates the counter of shareholders if necessary
    */
    function created(address _to, uint256 _value) public override onlyAgent {
        require(_value > 0, "No token created");
        updateShareholders(_to);
```

```
    }

   /**
    *  @dev See {ICompliance-destroyed}.
    *  updates the counter of shareholders if necessary
    */
    function destroyed(address _from, uint256 _value) public override onlyAgent {
        pruneShareholders(_from);
    }

   /**
    *  @dev See {ICompliance-transferOwnershipOnComplianceContract}.
    */
    function transferOwnershipOnComplianceContract(address newOwner) external
override onlyOwner {
        transferOwnership(newOwner);
    }
}
```

IClaimTopicsRegistry.sol

```
/**
 *     NOTICE
 *
 *     The T-REX software is licensed under a proprietary license or the GPL v.3.
 *     If you choose to receive it under the GPL v.3 license, the following applies:
 *     T-REX is a suite of smart contracts developed by Tokeny to manage and transfer
financial assets on the ethereum blockchain
 *
 *     Copyright (C) 2019, Tokeny sàrl.
 *
 *     This program is free software: you can redistribute it and/or modify
 *     it under the terms of the GNU General Public License as published by
 *     the Free Software Foundation, either version 3 of the License, or
 *     (at your option) any later version.
 *
 *     This program is distributed in the hope that it will be useful,
 *     but WITHOUT ANY WARRANTY; without even the implied warranty of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *     GNU General Public License for more details.
 *
 *     You should have received a copy of the GNU General Public License
 *     along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */

pragma solidity ^0.6.0;

interface IClaimTopicsRegistry {

   /**
    *  this event is emitted when a claim topic has been added to the
ClaimTopicsRegistry
    *  the event is emitted by the 'addClaimTopic' function
    *  `claimTopic` is the required claim added to the Claim Topics Registry
    */
    event ClaimTopicAdded(uint256 indexed claimTopic);
```

```
    /**
     *  this event is emitted when a claim topic has been removed from the
ClaimTopicsRegistry
     *  the event is emitted by the 'removeClaimTopic' function
     *  `claimTopic` is the required claim removed from the Claim Topics Registry
     */
     event ClaimTopicRemoved(uint256 indexed claimTopic);

    /**
     * @dev Add a trusted claim topic (For example: KYC=1, AML=2).
     * Only owner can call.
     * emits `ClaimTopicAdded` event
     * @param _claimTopic The claim topic index
     */
     function addClaimTopic(uint256 _claimTopic) external;

    /**
     *  @dev Remove a trusted claim topic (For example: KYC=1, AML=2).
     *  Only owner can call.
     *  emits `ClaimTopicRemoved` event
     *  @param _claimTopic The claim topic index
     */
     function removeClaimTopic(uint256 _claimTopic) external;

    /**
     *  @dev Get the trusted claim topics for the security token
     *  @return Array of trusted claim topics
     */
     function getClaimTopics() external view returns (uint256[] memory);

    /**
     *  @dev Transfers the Ownership of ClaimTopics to a new Owner.
     *  Only owner can call.
     *  @param _newOwner The new owner of this contract.
     */
     function transferOwnershipOnClaimTopicsRegistryContract(address _newOwner)
external;
}
```

IIdentityRegistry.sol

```
/**
 *     NOTICE
 *
 *     The T-REX software is licensed under a proprietary license or the GPL v.3.
 *     If you choose to receive it under the GPL v.3 license, the following applies:
 *     T-REX is a suite of smart contracts developed by Tokeny to manage and transfer
financial assets on the ethereum blockchain
 *
 *     Copyright (C) 2019, Tokeny sàrl.
 *
 *     This program is free software: you can redistribute it and/or modify
 *     it under the terms of the GNU General Public License as published by
 *     the Free Software Foundation, either version 3 of the License, or
 *     (at your option) any later version.
 *
 *     This program is distributed in the hope that it will be useful,
```

```
 *     but WITHOUT ANY WARRANTY; without even the implied warranty of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *     GNU General Public License for more details.
 *
 *     You should have received a copy of the GNU General Public License
 *     along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */

pragma solidity ^0.6.0;

import "../registry/ITrustedIssuersRegistry.sol";
import "../registry/IClaimTopicsRegistry.sol";
import "../registry/IIdentityRegistryStorage.sol";

import "@onchain-id/solidity/contracts/IClaimIssuer.sol";
import "@onchain-id/solidity/contracts/IIdentity.sol";

interface IIdentityRegistry {

   /**
    *  this event is emitted when the ClaimTopicsRegistry has been set for the
IdentityRegistry
    *  the event is emitted by the IdentityRegistry constructor
    *  `claimTopicsRegistry` is the address of the Claim Topics Registry contract
    */
    event ClaimTopicsRegistrySet(address indexed claimTopicsRegistry);

   /**
    *  this event is emitted when the IdentityRegistryStorage has been set for the
IdentityRegistry
    *  the event is emitted by the IdentityRegistry constructor
    *  `identityStorage` is the address of the Identity Registry Storage contract
    */
    event IdentityStorageSet(address indexed identityStorage);

   /**
    *  this event is emitted when the ClaimTopicsRegistry has been set for the
IdentityRegistry
    *  the event is emitted by the IdentityRegistry constructor
    *  `trustedIssuersRegistry` is the address of the Trusted Issuers Registry
contract
    */
    event TrustedIssuersRegistrySet(address indexed trustedIssuersRegistry);

   /**
    *  this event is emitted when an Identity is registered into the Identity
Registry.
    *  the event is emitted by the 'registerIdentity' function
    *  `investorAddress` is the address of the investor's wallet
    *  `identity` is the address of the Identity smart contract (onchainID)
    */
    event IdentityRegistered(address indexed investorAddress, IIdentity indexed
identity);

   /**
    *  this event is emitted when an Identity is removed from the Identity Registry.
    *  the event is emitted by the 'deleteIdentity' function
    *  `investorAddress` is the address of the investor's wallet
    *  `identity` is the address of the Identity smart contract (onchainID)
    */
```

```
    event IdentityRemoved(address indexed investorAddress, IIdentity indexed
identity);

   /**
    *  this event is emitted when an Identity has been updated
    *  the event is emitted by the 'updateIdentity' function
    *  `oldIdentity` is the old Identity contract's address to update
    *  `newIdentity` is the new Identity contract's
    */
    event IdentityUpdated(IIdentity indexed oldIdentity, IIdentity indexed
newIdentity);

   /**
    *  this event is emitted when an Identity's country has been updated
    *  the event is emitted by the 'updateCountry' function
    *  `investorAddress` is the address on which the country has been updated
    *  `country` is the numeric code (ISO 3166-1) of the new country
    */
    event CountryUpdated(address indexed investorAddress, uint16 indexed country);

   /**
    *  @dev Register an identity contract corresponding to a user address.
    *  Requires that the user doesn't have an identity contract already registered.
    *  This function can only be called by a wallet set as agent of the smart
contract
    *  @param _userAddress The address of the user
    *  @param _identity The address of the user's identity contract
    *  @param _country The country of the investor
    *  emits `IdentityRegistered` event
    */
    function registerIdentity(address _userAddress, IIdentity _identity, uint16
_country) external;

   /**
    *  @dev Removes an user from the identity registry.
    *  Requires that the user have an identity contract already deployed that will be
deleted.
    *  This function can only be called by a wallet set as agent of the smart
contract
    *  @param _userAddress The address of the user to be removed
    *  emits `IdentityRemoved` event
    */
    function deleteIdentity(address _userAddress) external;

   /**
    *  @dev Replace the actual identityRegistryStorage contract with a new one.
    *  This function can only be called by the wallet set as owner of the smart
contract
    *  @param _identityRegistryStorage The address of the new Identity Registry
Storage
    *  emits `IdentityStorageSet` event
    */
    function setIdentityRegistryStorage(address _identityRegistryStorage) external;

   /**
    *  @dev Replace the actual claimTopicsRegistry contract with a new one.
    *  This function can only be called by the wallet set as owner of the smart
contract
    *  @param _claimTopicsRegistry The address of the new claim Topics Registry
    *  emits `ClaimTopicsRegistrySet` event
```

```
    */
    function setClaimTopicsRegistry(address _claimTopicsRegistry) external;

   /**
    *  @dev Replace the actual trustedIssuersRegistry contract with a new one.
    *  This function can only be called by the wallet set as owner of the smart
contract
    *  @param _trustedIssuersRegistry The address of the new Trusted Issuers Registry
    *  emits `TrustedIssuersRegistrySet` event
    */
    function setTrustedIssuersRegistry(address _trustedIssuersRegistry) external;

   /**
    *  @dev Updates the country corresponding to a user address.
    *  Requires that the user should have an identity contract already deployed that
will be replaced.
    *  This function can only be called by a wallet set as agent of the smart
contract
    *  @param _userAddress The address of the user
    *  @param _country The new country of the user
    *  emits `CountryUpdated` event
    */
    function updateCountry(address _userAddress, uint16 _country) external;

   /**
    *  @dev Updates an identity contract corresponding to a user address.
    *  Requires that the user address should be the owner of the identity contract.
    *  Requires that the user should have an identity contract already deployed that
will be replaced.
    *  This function can only be called by a wallet set as agent of the smart
contract
    *  @param _userAddress The address of the user
    *  @param _identity The address of the user's new identity contract
    *  emits `IdentityUpdated` event
    */
    function updateIdentity(address _userAddress, IIdentity _identity) external;

   /**
    *  @dev function allowing to register identities in batch
    *  This function can only be called by a wallet set as agent of the smart
contract
    *  Requires that none of the users has an identity contract already registered.
    *  IMPORTANT : THIS TRANSACTION COULD EXCEED GAS LIMIT IF `_userAddresses.length`
IS TOO HIGH,
    *  USE WITH CARE OR YOU COULD LOSE TX FEES WITH AN "OUT OF GAS" TRANSACTION
    *  @param _userAddresses The addresses of the users
    *  @param _identities The addresses of the corresponding identity contracts
    *  @param _countries The countries of the corresponding investors
    *  emits _userAddresses.length `IdentityRegistered` events
    */
    function batchRegisterIdentity(address[] calldata _userAddresses, IIdentity[]
calldata _identities, uint16[] calldata _countries) external;

   /**
    *  @dev This functions checks whether a wallet has its Identity registered or not
    *  in the Identity Registry.
    *  @param _userAddress The address of the user to be checked.
    *  @return 'True' if the address is contained in the Identity Registry, 'false'
if not.
    */
```

```
    function contains(address _userAddress) external view returns (bool);

   /**
    *  @dev This functions checks whether an identity contract
    *  corresponding to the provided user address has the required claims or not
based
    *  on the data fetched from trusted issuers registry and from the claim topics
registry
    *  @param _userAddress The address of the user to be verified.
    *  @return 'True' if the address is verified, 'false' if not.
    */
    function isVerified(address _userAddress) external view returns (bool);

   /**
    *  @dev Returns the onchainID of an investor.
    *  @param _userAddress The wallet of the investor
    */
    function identity(address _userAddress) external view returns (IIdentity);

   /**
    *  @dev Returns the country code of an investor.
    *  @param _userAddress The wallet of the investor
    */
    function investorCountry(address _userAddress) external view returns (uint16);

   /**
    *  @dev Returns the IdentityRegistryStorage linked to the current
IdentityRegistry.
    */
    function identityStorage() external view returns (IIdentityRegistryStorage);

   /**
    *  @dev Returns the TrustedIssuersRegistry linked to the current
IdentityRegistry.
    */
    function issuersRegistry() external view returns (ITrustedIssuersRegistry);

   /**
    *  @dev Returns the ClaimTopicsRegistry linked to the current IdentityRegistry.
    */
    function topicsRegistry() external view returns (IClaimTopicsRegistry);

   /**
    *  @notice Transfers the Ownership of the Identity Registry to a new Owner.
    *  This function can only be called by the wallet set as owner of the smart
contract
    *  @param _newOwner The new owner of this contract.
    */
    function transferOwnershipOnIdentityRegistryContract(address _newOwner) external;

   /**
    *  @notice Adds an address as _agent of the Identity Registry Contract.
    *  This function can only be called by the wallet set as owner of the smart
contract
    *  @param _agent The _agent's address to add.
    */
    function addAgentOnIdentityRegistryContract(address _agent) external;

   /**
```

```
    *  @notice Removes an address from being _agent of the Identity Registry
Contract.
    *  This function can only be called by the wallet set as owner of the smart
contract
    *  @param _agent The _agent's address to remove.
    */
    function removeAgentOnIdentityRegistryContract(address _agent) external;
}
```

IIdentityRegistryStorage.sol

```
pragma solidity ^0.6.0;

import "@onchain-id/solidity/contracts/IIdentity.sol";

interface IIdentityRegistryStorage {

   /**
    *  this event is emitted when an Identity is registered into the storage
contract.
    *  the event is emitted by the 'registerIdentity' function
    *  `investorAddress` is the address of the investor's wallet
    *  `identity` is the address of the Identity smart contract (onchainID)
    */
    event IdentityStored(address indexed investorAddress, IIdentity indexed
identity);

   /**
    *  this event is emitted when an Identity is removed from the storage contract.
    *  the event is emitted by the 'deleteIdentity' function
    *  `investorAddress` is the address of the investor's wallet
    *  `identity` is the address of the Identity smart contract (onchainID)
    */
    event IdentityUnstored(address indexed investorAddress, IIdentity indexed
identity);

   /**
    *  this event is emitted when an Identity has been updated
    *  the event is emitted by the 'updateIdentity' function
    *  `oldIdentity` is the old Identity contract's address to update
    *  `newIdentity` is the new Identity contract's
    */
    event IdentityModified(IIdentity indexed oldIdentity, IIdentity indexed
newIdentity);

   /**
    *  this event is emitted when an Identity's country has been updated
    *  the event is emitted by the 'updateCountry' function
    *  `investorAddress` is the address on which the country has been updated
    *  `country` is the numeric code (ISO 3166-1) of the new country
    */
    event CountryModified(address indexed investorAddress, uint16 indexed country);

   /**
    *  this event is emitted when an Identity Registry is bound to the storage
contract
    *  the event is emitted by the 'addIdentityRegistry' function
```

```
     *  `identityRegistry` is the address of the identity registry added
     */
    event IdentityRegistryBound(address indexed identityRegistry);

   /**
    *  this event is emitted when an Identity Registry is unbound from the storage
contract
    *  the event is emitted by the 'removeIdentityRegistry' function
    *  `identityRegistry` is the address of the identity registry removed
    */
    event IdentityRegistryUnbound(address indexed identityRegistry);

   /**
    *  @dev Returns the identity registries linked to the storage contract
    */
    function linkedIdentityRegistries() external view returns (address[] memory);

   /**
    *  @dev Returns the onchainID of an investor.
    *  @param _userAddress The wallet of the investor
    */
    function storedIdentity(address _userAddress) external view returns (IIdentity);

   /**
    *  @dev Returns the country code of an investor.
    *  @param _userAddress The wallet of the investor
    */
    function storedInvestorCountry(address _userAddress) external view returns
(uint16);

   /**
    *  @dev adds an identity contract corresponding to a user address in the storage.
    *  Requires that the user doesn't have an identity contract already registered.
    *  This function can only be called by an address set as agent of the smart
contract
    *  @param _userAddress The address of the user
    *  @param _identity The address of the user's identity contract
    *  @param _country The country of the investor
    *  emits `IdentityStored` event
    */
    function addIdentityToStorage(address _userAddress, IIdentity _identity, uint16
_country) external;

   /**
    *  @dev Removes an user from the storage.
    *  Requires that the user have an identity contract already deployed that will be
deleted.
    *  This function can only be called by an address set as agent of the smart
contract
    *  @param _userAddress The address of the user to be removed
    *  emits `IdentityUnstored` event
    */
    function removeIdentityFromStorage(address _userAddress) external;

   /**
    *  @dev Updates the country corresponding to a user address.
    *  Requires that the user should have an identity contract already deployed that
will be replaced.
    *  This function can only be called by an address set as agent of the smart
contract
```

```
     *  @param _userAddress The address of the user
     *  @param _country The new country of the user
     *  emits `CountryModified` event
     */
    function modifyStoredInvestorCountry(address _userAddress, uint16 _country)
external;

   /**
     *  @dev Updates an identity contract corresponding to a user address.
     *  Requires that the user address should be the owner of the identity contract.
     *  Requires that the user should have an identity contract already deployed that
will be replaced.
     *  This function can only be called by an address set as agent of the smart
contract
     *  @param _userAddress The address of the user
     *  @param _identity The address of the user's new identity contract
     *  emits `IdentityModified` event
     */
    function modifyStoredIdentity(address _userAddress, IIdentity _identity)
external;

   /**
     *  @notice Transfers the Ownership of the Identity Registry Storage to a new
Owner.
     *  This function can only be called by the wallet set as owner of the smart
contract
     *  @param _newOwner The new owner of this contract.
     */
    function transferOwnershipOnIdentityRegistryStorage(address _newOwner) external;

   /**
     *  @notice Adds an identity registry as agent of the Identity Registry Storage
Contract.
     *  This function can only be called by the wallet set as owner of the smart
contract
     *  This function adds the identity registry to the list of identityRegistries
linked to the storage contract
     *  @param _identityRegistry The identity registry address to add.
     */
    function bindIdentityRegistry(address _identityRegistry) external;

   /**
     *  @notice Removes an identity registry from being agent of the Identity Registry
Storage Contract.
     *  This function can only be called by the wallet set as owner of the smart
contract
     *  This function removes the identity registry from the list of
identityRegistries linked to the storage contract
     *  @param _identityRegistry The identity registry address to remove.
     */
    function unbindIdentityRegistry(address _identityRegistry) external;
}
```

ITrustedIssuersRegistry.sol

```
/**
 *     NOTICE
```

```
 *
 *     The T-REX software is licensed under a proprietary license or the GPL v.3.
 *     If you choose to receive it under the GPL v.3 license, the following applies:
 *     T-REX is a suite of smart contracts developed by Tokeny to manage and transfer
financial assets on the ethereum blockchain
 *
 *     Copyright (C) 2019, Tokeny sàrl.
 *
 *     This program is free software: you can redistribute it and/or modify
 *     it under the terms of the GNU General Public License as published by
 *     the Free Software Foundation, either version 3 of the License, or
 *     (at your option) any later version.
 *
 *     This program is distributed in the hope that it will be useful,
 *     but WITHOUT ANY WARRANTY; without even the implied warranty of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *     GNU General Public License for more details.
 *
 *     You should have received a copy of the GNU General Public License
 *     along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */

pragma solidity ^0.6.0;

import "@onchain-id/solidity/contracts/IClaimIssuer.sol";

interface ITrustedIssuersRegistry {

    /**
     *  this event is emitted when a trusted issuer is added in the registry.
     *  the event is emitted by the addTrustedIssuer function
     *  `trustedIssuer` is the address of the trusted issuer's ClaimIssuer contract
     *  `claimTopics` is the set of claims that the trusted issuer is allowed to emit
     */
    event TrustedIssuerAdded(IClaimIssuer indexed trustedIssuer, uint[] claimTopics);

    /**
     *  this event is emitted when a trusted issuer is removed from the registry.
     *  the event is emitted by the removeTrustedIssuer function
     *  `trustedIssuer` is the address of the trusted issuer's ClaimIssuer contract
     */
    event TrustedIssuerRemoved(IClaimIssuer indexed trustedIssuer);

    /**
     *  this event is emitted when the set of claim topics is changed for a given
trusted issuer.
     *  the event is emitted by the updateIssuerClaimTopics function
     *  `trustedIssuer` is the address of the trusted issuer's ClaimIssuer contract
     *  `claimTopics` is the set of claims that the trusted issuer is allowed to emit
     */
    event ClaimTopicsUpdated(IClaimIssuer indexed trustedIssuer, uint[] claimTopics);

    /**
     *  @dev registers a ClaimIssuer contract as trusted claim issuer.
     *  Requires that a ClaimIssuer contract doesn't already exist
     *  Requires that the claimTopics set is not empty
     *  @param _trustedIssuer The ClaimIssuer contract address of the trusted claim
issuer.
     *  @param _claimTopics the set of claim topics that the trusted issuer is allowed
to emit
```

```
    *  This function can only be called by the owner of the Trusted Issuers Registry
contract
    *  emits a `TrustedIssuerAdded` event
    */
    function addTrustedIssuer(IClaimIssuer _trustedIssuer, uint[] calldata
_claimTopics) external;

   /**
    *  @dev Removes the ClaimIssuer contract of a trusted claim issuer.
    *  Requires that the claim issuer contract to be registered first
    *  @param _trustedIssuer the claim issuer to remove.
    *  This function can only be called by the owner of the Trusted Issuers Registry
contract
    *  emits a `TrustedIssuerRemoved` event
    */
    function removeTrustedIssuer(IClaimIssuer _trustedIssuer) external;

   /**
    *  @dev Updates the set of claim topics that a trusted issuer is allowed to emit.
    *  Requires that this ClaimIssuer contract already exists in the registry
    *  Requires that the provided claimTopics set is not empty
    *  @param _trustedIssuer the claim issuer to update.
    *  @param _claimTopics the set of claim topics that the trusted issuer is allowed
to emit
    *  This function can only be called by the owner of the Trusted Issuers Registry
contract
    *  emits a `ClaimTopicsUpdated` event
    */
    function updateIssuerClaimTopics(IClaimIssuer _trustedIssuer, uint[] calldata
_claimTopics) external;

   /**
    *  @dev Function for getting all the trusted claim issuers stored.
    *  @return array of all claim issuers registered.
    */
    function getTrustedIssuers() external view returns (IClaimIssuer[] memory);

   /**
    *  @dev Checks if the ClaimIssuer contract is trusted
    *  @param _issuer the address of the ClaimIssuer contract
    *  @return true if the issuer is trusted, false otherwise.
    */
    function isTrustedIssuer(address _issuer) external view returns(bool);

   /**
    *  @dev Function for getting all the claim topic of trusted claim issuer
    *  Requires the provided ClaimIssuer contract to be registered in the trusted
issuers registry.
    *  @param _trustedIssuer the trusted issuer concerned.
    *  @return The set of claim topics that the trusted issuer is allowed to emit
    */
    function getTrustedIssuerClaimTopics(IClaimIssuer _trustedIssuer) external view
returns(uint[] memory);

   /**
    *  @dev Function for checking if the trusted claim issuer is allowed
    *  to emit a certain claim topic
    *  @param _issuer the address of the trusted issuer's ClaimIssuer contract
    *  @param _claimTopic the Claim Topic that has to be checked to know if the
`issuer` is allowed to emit it
```

```
    *  @return true if the issuer is trusted for this claim topic.
    */
    function hasClaimTopic(address _issuer, uint _claimTopic) external view
returns(bool);

   /**
    *  @dev Transfers the Ownership of TrustedIssuersRegistry to a new Owner.
    *  @param _newOwner The new owner of this contract.
    *  This function can only be called by the owner of the Trusted Issuers Registry
contract
    *  emits an `OwnershipTransferred` event
    */
    function transferOwnershipOnIssuersRegistryContract(address _newOwner) external;
}
```

IdentityRegistry.sol

```
/**
 *     NOTICE
 *
 *     The T-REX software is licensed under a proprietary license or the GPL v.3.
 *     If you choose to receive it under the GPL v.3 license, the following applies:
 *     T-REX is a suite of smart contracts developed by Tokeny to manage and transfer
financial assets on the ethereum blockchain
 *
 *     Copyright (C) 2019, Tokeny sàrl.
 *
 *     This program is free software: you can redistribute it and/or modify
 *     it under the terms of the GNU General Public License as published by
 *     the Free Software Foundation, either version 3 of the License, or
 *     (at your option) any later version.
 *
 *     This program is distributed in the hope that it will be useful,
 *     but WITHOUT ANY WARRANTY; without even the implied warranty of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *     GNU General Public License for more details.
 *
 *     You should have received a copy of the GNU General Public License
 *     along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */

pragma solidity ^0.6.0;


import "@onchain-id/solidity/contracts/IClaimIssuer.sol";
import "@onchain-id/solidity/contracts/IIdentity.sol";
import "../registry/IClaimTopicsRegistry.sol";
import "../registry/ITrustedIssuersRegistry.sol";
import "../registry/IIdentityRegistry.sol";
import "../roles/AgentRole.sol";
import "../registry/IIdentityRegistryStorage.sol";
import "../roles/Ownable.sol";

contract IdentityRegistry is IIdentityRegistry, AgentRole {


    /// Address of the ClaimTopicsRegistry Contract
```

```
    IClaimTopicsRegistry private tokenTopicsRegistry;

    /// Address of the TrustedIssuersRegistry Contract
    ITrustedIssuersRegistry private tokenIssuersRegistry;

    /// Address of the IdentityRegistryStorage Contract
    IIdentityRegistryStorage private tokenIdentityStorage;

   /**
    *  @dev the constructor initiates the Identity Registry smart contract
    *  @param _trustedIssuersRegistry the trusted issuers registry linked to the
Identity Registry
    *  @param _claimTopicsRegistry the claim topics registry linked to the Identity
Registry
    *  @param _identityStorage the identity registry storage linked to the Identity
Registry
    *  emits a `ClaimTopicsRegistrySet` event
    *  emits a `TrustedIssuersRegistrySet` event
    *  emits an `IdentityStorageSet` event
    */
    constructor (
        address _trustedIssuersRegistry,
        address _claimTopicsRegistry,
        address _identityStorage
    ) public {
        tokenTopicsRegistry = IClaimTopicsRegistry(_claimTopicsRegistry);
        tokenIssuersRegistry = ITrustedIssuersRegistry(_trustedIssuersRegistry);
        tokenIdentityStorage = IIdentityRegistryStorage(_identityStorage);
        emit ClaimTopicsRegistrySet(_claimTopicsRegistry);
        emit TrustedIssuersRegistrySet(_trustedIssuersRegistry);
        emit IdentityStorageSet(_identityStorage);
    }

   /**
    *  @dev See {IIdentityRegistry-identity}.
    */
    function identity(address _userAddress) public override view returns (IIdentity){
        return tokenIdentityStorage.storedIdentity(_userAddress);
    }

   /**
    *  @dev See {IIdentityRegistry-investorCountry}.
    */
    function investorCountry(address _userAddress) public override view returns
(uint16){
        return tokenIdentityStorage.storedInvestorCountry(_userAddress);
    }

   /**
    *  @dev See {IIdentityRegistry-issuersRegistry}.
    */
    function issuersRegistry() public override view returns
(ITrustedIssuersRegistry){
        return tokenIssuersRegistry;
    }

   /**
    *  @dev See {IIdentityRegistry-topicsRegistry}.
    */
    function topicsRegistry() public override view returns (IClaimTopicsRegistry){
```

```
        return tokenTopicsRegistry;
    }

    /**
    *  @dev See {IIdentityRegistry-identityStorage}.
    */
    function identityStorage() public override view returns
(IIdentityRegistryStorage){
        return tokenIdentityStorage;
    }

   /**
    *  @dev See {IIdentityRegistry-registerIdentity}.
    */
    function registerIdentity(address _userAddress, IIdentity _identity, uint16
_country) public override onlyAgent {
        tokenIdentityStorage.addIdentityToStorage(_userAddress, _identity, _country);
        emit IdentityRegistered(_userAddress, _identity);
    }

   /**
    *  @dev See {IIdentityRegistry-batchRegisterIdentity}.
    */
    function batchRegisterIdentity(address[] calldata _userAddresses, IIdentity[]
calldata _identities, uint16[] calldata _countries) external override {
        for (uint256 i = 0; i < _userAddresses.length; i++) {
            registerIdentity(_userAddresses[i], _identities[i], _countries[i]);
        }
    }

   /**
    *  @dev See {IIdentityRegistry-updateIdentity}.
    */
    function updateIdentity(address _userAddress, IIdentity _identity) public
override onlyAgent {
        tokenIdentityStorage.modifyStoredIdentity(_userAddress, _identity);
        emit IdentityUpdated(identity(_userAddress), _identity);
    }


   /**
    *  @dev See {IIdentityRegistry-updateCountry}.
    */
    function updateCountry(address _userAddress, uint16 _country) public override
onlyAgent {
        tokenIdentityStorage.modifyStoredInvestorCountry(_userAddress, _country);
        emit CountryUpdated(_userAddress, _country);
    }

   /**
    *  @dev See {IIdentityRegistry-deleteIdentity}.
    */
    function deleteIdentity(address _userAddress) public override onlyAgent {
        tokenIdentityStorage.removeIdentityFromStorage(_userAddress);
        emit IdentityRemoved(_userAddress, identity(_userAddress));
    }

   /**
    *  @dev See {IIdentityRegistry-isVerified}.
    */
```

```
    function isVerified(address _userAddress) public override view returns (bool) {
        if (address(identity(_userAddress)) == address(0)) {
            return false;
        }
        uint256[] memory claimTopics = tokenTopicsRegistry.getClaimTopics();
        uint length = claimTopics.length;
        if (length == 0) {
            return true;
        }
        uint256 foundClaimTopic;
        uint256 scheme;
        address issuer;
        bytes memory sig;
        bytes memory data;
        uint256 claimTopic;
        for (claimTopic = 0; claimTopic < length; claimTopic++) {
            bytes32[] memory claimIds =
identity(_userAddress).getClaimIdsByTopic(claimTopics[claimTopic]);
            if (claimIds.length == 0) {
                return false;
            }
            for (uint j = 0; j < claimIds.length; j++) {
                (foundClaimTopic, scheme, issuer, sig, data,) =
identity(_userAddress).getClaim(claimIds[j]);
                if (!tokenIssuersRegistry.isTrustedIssuer(issuer)) {
                    return false;
                }
                if (!tokenIssuersRegistry.hasClaimTopic(issuer,
claimTopics[claimTopic])) {
                    return false;
                }
                if (!IClaimIssuer(issuer).isClaimValid(identity(_userAddress),
claimTopics[claimTopic], sig, data)) {
                    return false;
                }
            }
        }
        return true;
    }

   /**
    *  @dev See {IIdentityRegistry-setIdentityRegistryStorage}.
    */
    function setIdentityRegistryStorage(address _identityRegistryStorage) public
override onlyOwner {
        tokenIdentityStorage = IIdentityRegistryStorage(_identityRegistryStorage);
        emit IdentityStorageSet(_identityRegistryStorage);
    }

   /**
    *  @dev See {IIdentityRegistry-setClaimTopicsRegistry}.
    */
    function setClaimTopicsRegistry(address _claimTopicsRegistry) public override
onlyOwner {
        tokenTopicsRegistry = IClaimTopicsRegistry(_claimTopicsRegistry);
        emit ClaimTopicsRegistrySet(_claimTopicsRegistry);
    }

   /**
    *  @dev See {IIdentityRegistry-setTrustedIssuersRegistry}.
```

```
    */
    function setTrustedIssuersRegistry(address _trustedIssuersRegistry) public
override onlyOwner {
        tokenIssuersRegistry = ITrustedIssuersRegistry(_trustedIssuersRegistry);
        emit TrustedIssuersRegistrySet(_trustedIssuersRegistry);
    }

   /**
    *  @dev See {IIdentityRegistry-contains}.
    */
    function contains(address _userAddress) public override view returns (bool){
        if (address(identity(_userAddress)) == address(0)) {
            return false;
        }
        return true;
    }

   /**
    *  @dev See {IIdentityRegistry-transferOwnershipOnIdentityRegistryContract}.
    */
    function transferOwnershipOnIdentityRegistryContract(address _newOwner) external
override onlyOwner {
        transferOwnership(_newOwner);
    }

   /**
    *  @dev See {IIdentityRegistry-addAgentOnIdentityRegistryContract}.
    */
    function addAgentOnIdentityRegistryContract(address _agent) external override {
        addAgent(_agent);
    }

   /**
    *  @dev See {IIdentityRegistry-removeAgentOnIdentityRegistryContract}.
    */
    function removeAgentOnIdentityRegistryContract(address _agent) external override
{
        removeAgent(_agent);
    }
}
```

IdentityRegistry.sol

```
pragma solidity ^0.6.0;

import "@onchain-id/solidity/contracts/IIdentity.sol";
import "../roles/AgentRole.sol";
import "../registry/IIdentityRegistryStorage.sol";


contract IdentityRegistryStorage is IIdentityRegistryStorage, AgentRole {

    /// struct containing the identity contract and the country of the user
    struct Identity {
        IIdentity identityContract;
        uint16 investorCountry;
    }
```

```
    /// mapping between a user address and the corresponding identity
    mapping(address => IIdentity) private identities;

    /// array of Identity Registries linked to this storage
    address[] private identityRegistries;

   /**
    *  @dev See {IIdentityRegistryStorage-linkedIdentityRegistries}.
    */
    function linkedIdentityRegistries() public override view returns (address[]
memory){
        return identityRegistries;
    }

   /**
    *  @dev See {IIdentityRegistryStorage-storedIdentity}.
    */
    function storedIdentity(address _userAddress) public override view returns
(IIdentity){
        return identities[_userAddress].identityContract;
    }

   /**
    *  @dev See {IIdentityRegistryStorage-storedInvestorCountry}.
    */
    function storedInvestorCountry(address _userAddress) public override view returns
(uint16){
        return identities[_userAddress].investorCountry;
    }

   /**
    *  @dev See {IIdentityRegistryStorage-addIdentityToStorage}.
    */
    function addIdentityToStorage(address _userAddress, IIdentity _identity, uint16
_country) public override onlyAgent {
        require(address(_identity) != address(0), "contract address can't be a zero
address");
        require(address(identities[_userAddress].identityContract) == address(0),
"identity contract already exists, please use update");
        identities[_userAddress].identityContract = _identity;
        identities[_userAddress].investorCountry = _country;
        emit IdentityStored(_userAddress, _identity);
    }

   /**
    *  @dev See {IIdentityRegistryStorage-modifyStoredIdentity}.
    */
    function modifyStoredIdentity(address _userAddress, IIdentity _identity) public
override onlyAgent {
        require(address(identities[_userAddress].identityContract) != address(0),
"this user has no identity registered");
        require(address(_identity) != address(0), "contract address can't be a zero
address");
        identities[_userAddress].identityContract = _identity;
        emit IdentityModified(identities[_userAddress].identityContract, _identity);
    }

   /**
    *  @dev See {IIdentityRegistryStorage-modifyStoredInvestorCountry}.
```

```
    */
    function modifyStoredInvestorCountry(address _userAddress, uint16 _country)
public override onlyAgent {
        require(address(identities[_userAddress].identityContract) != address(0),
"this user has no identity registered");
        identities[_userAddress].investorCountry = _country;
        emit CountryModified(_userAddress, _country);
    }

   /**
    *  @dev See {IIdentityRegistryStorage-removeIdentityFromStorage}.
    */
    function removeIdentityFromStorage(address _userAddress) public override
onlyAgent {
        require(address(identities[_userAddress].identityContract) != address(0),
"you haven't registered an identity yet");
        delete identities[_userAddress];
        emit IdentityUnstored(_userAddress,
identities[_userAddress].identityContract);
    }

   /**
    *  @dev See {IIdentityRegistryStorage-
transferOwnershipOnIdentityRegistryStorage}.
    */
    function transferOwnershipOnIdentityRegistryStorage(address _newOwner) external
override onlyOwner {
        transferOwnership(_newOwner);
    }

    /**
    *  @dev See {IIdentityRegistryStorage-bindIdentityRegistry}.
    */
    function bindIdentityRegistry(address _identityRegistry) external override {
        addAgent(_identityRegistry);
        identityRegistries.push(_identityRegistry);
        emit IdentityRegistryBound(_identityRegistry);
    }

    /**
     *  @dev See {IIdentityRegistryStorage-unbindIdentityRegistry}.
     */
    function unbindIdentityRegistry(address _identityRegistry) external override {
        require(identityRegistries.length > 0, "identity registry is not stored");
        uint length = identityRegistries.length;
        for (uint i = 0; i < length; i++) {
            if (identityRegistries[i] == _identityRegistry) {
                delete identityRegistries[i];
                identityRegistries[i] = identityRegistries[length - 1];
                delete identityRegistries[length - 1];
                identityRegistries.pop();
                break;
            }
        }
        removeAgent(_identityRegistry);
        emit IdentityRegistryUnbound(_identityRegistry);
    }
}
```

kaspersky

TrustedIssuersRegistry.sol

```
/**
 *     NOTICE
 *
 *     The T-REX software is licensed under a proprietary license or the GPL v.3.
 *     If you choose to receive it under the GPL v.3 license, the following applies:
 *     T-REX is a suite of smart contracts developed by Tokeny to manage and transfer
financial assets on the ethereum blockchain
 *
 *     Copyright (C) 2019, Tokeny sàrl.
 *
 *     This program is free software: you can redistribute it and/or modify
 *     it under the terms of the GNU General Public License as published by
 *     the Free Software Foundation, either version 3 of the License, or
 *     (at your option) any later version.
 *
 *     This program is distributed in the hope that it will be useful,
 *     but WITHOUT ANY WARRANTY; without even the implied warranty of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *     GNU General Public License for more details.
 *
 *     You should have received a copy of the GNU General Public License
 *     along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */

pragma solidity ^0.6.0;

import "@onchain-id/solidity/contracts/IClaimIssuer.sol";

import "../registry/ITrustedIssuersRegistry.sol";
import "../roles/Ownable.sol";

contract TrustedIssuersRegistry is ITrustedIssuersRegistry, Ownable {

    /// Array containing all TrustedIssuers identity contract address.
    IClaimIssuer[] private trustedIssuers;

    /// Mapping between a trusted issuer index and its corresponding claimTopics.
    mapping(address => uint[]) private trustedIssuerClaimTopics;

   /**
    *  @dev See {ITrustedIssuersRegistry-addTrustedIssuer}.
    */
    function addTrustedIssuer(IClaimIssuer _trustedIssuer, uint[] memory
_claimTopics) public override onlyOwner {
        require(trustedIssuerClaimTopics[address(_trustedIssuer)].length == 0,
"trusted Issuer already exists");
        require(_claimTopics.length > 0, "trusted claim topics cannot be empty");
        trustedIssuers.push(_trustedIssuer);
        trustedIssuerClaimTopics[address(_trustedIssuer)] = _claimTopics;
        emit TrustedIssuerAdded(_trustedIssuer, _claimTopics);
    }

   /**
    *  @dev See {ITrustedIssuersRegistry-removeTrustedIssuer}.
    */
    function removeTrustedIssuer(IClaimIssuer _trustedIssuer) public override
onlyOwner {
```

```
        require(trustedIssuerClaimTopics[address(_trustedIssuer)].length != 0,
"trusted Issuer doesn't exist");
        uint length = trustedIssuers.length;
        for (uint i = 0; i < length; i++) {
            if (trustedIssuers[i] == _trustedIssuer) {
                delete trustedIssuers[i];
                trustedIssuers[i] = trustedIssuers[length - 1];
                delete trustedIssuers[length - 1];
                trustedIssuers.pop();
                break;
            }
        }
        delete trustedIssuerClaimTopics[address(_trustedIssuer)];
        emit TrustedIssuerRemoved(_trustedIssuer);
    }

   /**
    *  @dev See {ITrustedIssuersRegistry-updateIssuerClaimTopics}.
    */
    function updateIssuerClaimTopics(IClaimIssuer _trustedIssuer, uint[] memory
_claimTopics) public override onlyOwner {
        require(trustedIssuerClaimTopics[address(_trustedIssuer)].length != 0,
"trusted Issuer doesn't exist");
        require(_claimTopics.length > 0, "claim topics cannot be empty");
        trustedIssuerClaimTopics[address(_trustedIssuer)] = _claimTopics;
        emit ClaimTopicsUpdated(_trustedIssuer, _claimTopics);
    }

   /**
    *  @dev See {ITrustedIssuersRegistry-getTrustedIssuers}.
    */
    function getTrustedIssuers() public override view returns (IClaimIssuer[] memory)
{
        return trustedIssuers;
    }

   /**
    *  @dev See {ITrustedIssuersRegistry-isTrustedIssuer}.
    */
    function isTrustedIssuer(address _issuer) public override view returns (bool) {
        uint length = trustedIssuers.length;
        for (uint i = 0; i < length; i++) {
            if (address(trustedIssuers[i]) == _issuer) {
                return true;
            }
        }
        return false;
    }

   /**
    *  @dev See {ITrustedIssuersRegistry-getTrustedIssuerClaimTopics}.
    */
    function getTrustedIssuerClaimTopics(IClaimIssuer _trustedIssuer) public override
view returns (uint[] memory) {
        require(trustedIssuerClaimTopics[address(_trustedIssuer)].length != 0,
"trusted Issuer doesn't exist");
        return trustedIssuerClaimTopics[address(_trustedIssuer)];
    }

   /**
```

```
    *  @dev See {ITrustedIssuersRegistry-hasClaimTopic}.
    */
    function hasClaimTopic(address _issuer, uint _claimTopic) public override view
returns (bool) {
        uint length = trustedIssuerClaimTopics[_issuer].length;
        uint[] memory claimTopics = trustedIssuerClaimTopics[_issuer];
        for (uint i = 0; i < length; i++) {
            if (claimTopics[i] == _claimTopic) {
                return true;
            }
        }
        return false;
    }

   /**
    *  @dev See {ITrustedIssuersRegistry-transferOwnershipOnIssuersRegistryContract}.
    */
    function transferOwnershipOnIssuersRegistryContract(address _newOwner) external
override onlyOwner {
        transferOwnership(_newOwner);
    }
}
```

ClaimTopicsRegistry.sol

```
/**
 *     NOTICE
 *
 *     The T-REX software is licensed under a proprietary license or the GPL v.3.
 *     If you choose to receive it under the GPL v.3 license, the following applies:
 *     T-REX is a suite of smart contracts developed by Tokeny to manage and transfer
financial assets on the ethereum blockchain
 *
 *     Copyright (C) 2019, Tokeny sàrl.
 *
 *     This program is free software: you can redistribute it and/or modify
 *     it under the terms of the GNU General Public License as published by
 *     the Free Software Foundation, either version 3 of the License, or
 *     (at your option) any later version.
 *
 *     This program is distributed in the hope that it will be useful,
 *     but WITHOUT ANY WARRANTY; without even the implied warranty of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *     GNU General Public License for more details.
 *
 *     You should have received a copy of the GNU General Public License
 *     along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */

pragma solidity ^0.6.0;

import "../registry/IClaimTopicsRegistry.sol";
import "../roles/Ownable.sol";

contract ClaimTopicsRegistry is IClaimTopicsRegistry, Ownable {

    /// All required Claim Topics
```

```
    uint256[] private claimTopics;

    /**
     *  @dev See {IClaimTopicsRegistry-addClaimTopic}.
     */
    function addClaimTopic(uint256 _claimTopic) public override onlyOwner {
        uint length = claimTopics.length;
        for (uint i = 0; i < length; i++) {
            require(claimTopics[i] != _claimTopic, "claimTopic already exists");
        }
        claimTopics.push(_claimTopic);
        emit ClaimTopicAdded(_claimTopic);
    }

    /**
     *  @dev See {IClaimTopicsRegistry-removeClaimTopic}.
     */
    function removeClaimTopic(uint256 _claimTopic) public override onlyOwner {
        uint length = claimTopics.length;
        for (uint i = 0; i < length; i++) {
            if (claimTopics[i] == _claimTopic) {
                delete claimTopics[i];
                claimTopics[i] = claimTopics[length - 1];
                delete claimTopics[length - 1];
                claimTopics.pop();
                emit ClaimTopicRemoved(_claimTopic);
                break;
            }
        }
    }

    /**
     *  @dev See {IClaimTopicsRegistry-getClaimTopics}.
     */
    function getClaimTopics() public override view returns (uint256[] memory) {
        return claimTopics;
    }

    /**
     *  @dev See {IClaimTopicsRegistry-
transferOwnershipOnClaimTopicsRegistryContract}.
     */
    function transferOwnershipOnClaimTopicsRegistryContract(address _newOwner)
external override onlyOwner {
        transferOwnership(_newOwner);
    }
}
```

AgentManager.sol

```
/**
 *     NOTICE
 *
 *     The T-REX software is licensed under a proprietary license or the GPL v.3.
 *     If you choose to receive it under the GPL v.3 license, the following applies:
 *     T-REX is a suite of smart contracts developed by Tokeny to manage and transfer
financial assets on the ethereum blockchain
```

```
 *
 *     Copyright (C) 2019, Tokeny sàrl.
 *
 *     This program is free software: you can redistribute it and/or modify
 *     it under the terms of the GNU General Public License as published by
 *     the Free Software Foundation, either version 3 of the License, or
 *     (at your option) any later version.
 *
 *     This program is distributed in the hope that it will be useful,
 *     but WITHOUT ANY WARRANTY; without even the implied warranty of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *     GNU General Public License for more details.
 *
 *     You should have received a copy of the GNU General Public License
 *     along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */

pragma solidity ^0.6.0;

import "../token/IToken.sol";
import "../registry/IIdentityRegistry.sol";
import "./AgentRoles.sol";
import "@onchain-id/solidity/contracts/IIdentity.sol";

contract AgentManager is AgentRoles {

    /// the token managed by this AgentManager contract
    IToken public token;

    constructor (address _token) public {
        token = IToken(_token);
    }

   /**
    *  @dev calls the `forcedTransfer` function on the Token contract
    *  AgentManager has to be set as agent on the token smart contract to process
this function
    *  See {IToken-forcedTransfer}.
    *  Requires that `_onchainID` is set as TransferManager on the AgentManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callForcedTransfer(address _from, address _to, uint256 _amount,
IIdentity _onchainID) external {
        require(isTransferManager(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Transfer Manager");
        token.forcedTransfer(_from, _to, _amount);
    }

   /**
    *  @dev calls the `batchForcedTransfer` function on the Token contract
    *  AgentManager has to be set as agent on the token smart contract to process
this function
    *  See {IToken-batchForcedTransfer}.
    *  Requires that `_onchainID` is set as TransferManager on the AgentManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
```

```
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callBatchForcedTransfer(address[] calldata _fromList, address[] calldata
_toList, uint256[] calldata _amounts, IIdentity _onchainID) external {
        require(isTransferManager(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Transfer Manager");
        token.batchForcedTransfer(_fromList, _toList, _amounts);
    }

   /**
    *  @dev calls the `pause` function on the Token contract
    *  AgentManager has to be set as agent on the token smart contract to process
this function
    *  See {IToken-pause}.
    *  Requires that `_onchainID` is set as Freezer on the AgentManager contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callPause(IIdentity _onchainID) external {
        require(isFreezer(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Freezer");
        token.pause();
    }

   /**
    *  @dev calls the `unpause` function on the Token contract
    *  AgentManager has to be set as agent on the token smart contract to process
this function
    *  See {IToken-unpause}.
    *  Requires that `_onchainID` is set as Freezer on the AgentManager contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callUnpause(IIdentity _onchainID) external {
        require(isFreezer(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Freezer");
        token.unpause();
    }

   /**
    *  @dev calls the `mint` function on the Token contract
    *  AgentManager has to be set as agent on the token smart contract to process
this function
    *  See {IToken-mint}.
    *  Requires that `_onchainID` is set as SupplyModifier on the AgentManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callMint(address _to, uint256 _amount, IIdentity _onchainID) external {
        require(isSupplyModifier(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Supply Modifier");
```

```
        token.mint(_to, _amount);
    }

   /**
    *  @dev calls the `batchMint` function on the Token contract
    *  AgentManager has to be set as agent on the token smart contract to process
this function
    *  See {IToken-batchMint}.
    *  Requires that `_onchainID` is set as SupplyModifier on the AgentManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callBatchMint(address[] calldata _toList, uint256[] calldata _amounts,
IIdentity _onchainID) external {
        require(isSupplyModifier(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Supply Modifier");
        token.batchMint(_toList, _amounts);
    }

   /**
    *  @dev calls the `burn` function on the Token contract
    *  AgentManager has to be set as agent on the token smart contract to process
this function
    *  See {IToken-burn}.
    *  Requires that `_onchainID` is set as SupplyModifier on the AgentManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callBurn(address _userAddress, uint256 _amount, IIdentity _onchainID)
external {
        require(isSupplyModifier(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Supply Modifier");
        token.burn(_userAddress, _amount);
    }

   /**
    *  @dev calls the `batchBurn` function on the Token contract
    *  AgentManager has to be set as agent on the token smart contract to process
this function
    *  See {IToken-batchBurn}.
    *  Requires that `_onchainID` is set as SupplyModifier on the AgentManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callBatchBurn(address[] calldata _userAddresses, uint256[] calldata
_amounts, IIdentity _onchainID) external {
        require(isSupplyModifier(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Supply Modifier");
        token.batchBurn(_userAddresses, _amounts);
    }
```

```
   /**
    *  @dev calls the `setAddressFrozen` function on the Token contract
    *  AgentManager has to be set as agent on the token smart contract to process
this function
    *  See {IToken-setAddressFrozen}.
    *  Requires that `_onchainID` is set as Freezer on the AgentManager contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callSetAddressFrozen(address _userAddress, bool _freeze, IIdentity
_onchainID) external {
        require(isFreezer(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Freezer");
        token.setAddressFrozen(_userAddress, _freeze);
    }

   /**
    *  @dev calls the `batchSetAddressFrozen` function on the Token contract
    *  AgentManager has to be set as agent on the token smart contract to process
this function
    *  See {IToken-batchSetAddressFrozen}.
    *  Requires that `_onchainID` is set as Freezer on the AgentManager contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callBatchSetAddressFrozen(address[] calldata _userAddresses, bool[]
calldata _freeze, IIdentity _onchainID) external {
        require(isFreezer(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Freezer");
        token.batchSetAddressFrozen(_userAddresses, _freeze);
    }

   /**
    *  @dev calls the `freezePartialTokens` function on the Token contract
    *  AgentManager has to be set as agent on the token smart contract to process
this function
    *  See {IToken-freezePartialTokens}.
    *  Requires that `_onchainID` is set as Freezer on the AgentManager contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callFreezePartialTokens(address _userAddress, uint256 _amount, IIdentity
_onchainID) external {
        require(isFreezer(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Freezer");
        token.freezePartialTokens(_userAddress, _amount);
    }

   /**
    *  @dev calls the `batchFreezePartialTokens` function on the Token contract
    *  AgentManager has to be set as agent on the token smart contract to process
this function
    *  See {IToken-batchFreezePartialTokens}.
    *  Requires that `_onchainID` is set as Freezer on the AgentManager contract
```

```
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callBatchFreezePartialTokens(address[] calldata _userAddresses,
uint256[] calldata _amounts, IIdentity _onchainID) external {
        require(isFreezer(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Freezer");
        token.batchFreezePartialTokens(_userAddresses, _amounts);
    }

   /**
    *  @dev calls the `unfreezePartialTokens` function on the Token contract
    *  AgentManager has to be set as agent on the token smart contract to process
this function
    *  See {IToken-unfreezePartialTokens}.
    *  Requires that `_onchainID` is set as Freezer on the AgentManager contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callUnfreezePartialTokens(address _userAddress, uint256 _amount,
IIdentity _onchainID) external {
        require(isFreezer(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Freezer");
        token.unfreezePartialTokens(_userAddress, _amount);
    }

   /**
    *  @dev calls the `batchUnfreezePartialTokens` function on the Token contract
    *  AgentManager has to be set as agent on the token smart contract to process
this function
    *  See {IToken-batchUnfreezePartialTokens}.
    *  Requires that `_onchainID` is set as Freezer on the AgentManager contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callBatchUnfreezePartialTokens(address[] calldata _userAddresses,
uint256[] calldata _amounts, IIdentity _onchainID) external {
        require(isFreezer(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Freezer");
        token.batchUnfreezePartialTokens(_userAddresses, _amounts);
    }

   /**
    *  @dev calls the `recoveryAddress` function on the Token contract
    *  AgentManager has to be set as agent on the token smart contract to process
this function
    *  See {IToken-recoveryAddress}.
    *  Requires that `_managerOnchainID` is set as RecoveryAgent on the AgentManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_managerOnchainID`
    *  @param _managerOnchainID the onchainID contract of the caller, e.g. "i call
this function and i am Bob"
    */
```

```
    function callRecoveryAddress(address _lostWallet, address _newWallet, address
_onchainID, IIdentity _managerOnchainID) external {
        require(isRecoveryAgent(address(_managerOnchainID)) &&
_managerOnchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender
is NOT Recovery Agent");
        token.recoveryAddress(_lostWallet, _newWallet, _onchainID);
    }

   /**
    *  @dev calls the `registerIdentity` function on the Identity Registry contract
    *  AgentManager has to be set as agent on the Identity Registry smart contract to
process this function
    *  See {IIdentityRegistry-registerIdentity}.
    *  Requires that `ManagerOnchainID` is set as WhiteListManager on the
AgentManager contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_managerOnchainID`
    *  @param _managerOnchainID the onchainID contract of the caller, e.g. "i call
this function and i am Bob"
    */
    function callRegisterIdentity(address _userAddress, IIdentity _onchainID, uint16
_country, IIdentity _managerOnchainID) external {
        require(isWhiteListManager(address(_managerOnchainID)) &&
_managerOnchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender
is NOT WhiteList Manager");
        token.identityRegistry().registerIdentity(_userAddress, _onchainID,
_country);
    }

   /**
    *  @dev calls the `updateIdentity` function on the Identity Registry contract
    *  AgentManager has to be set as agent on the Identity Registry smart contract to
process this function
    *  See {IIdentityRegistry-updateIdentity}.
    *  Requires that `_onchainID` is set as WhiteListManager on the AgentManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callUpdateIdentity(address _userAddress, IIdentity _identity, IIdentity
_onchainID) external {
        require(isWhiteListManager(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
WhiteList Manager");
        token.identityRegistry().updateIdentity(_userAddress, _identity);
    }

   /**
    *  @dev calls the `updateCountry` function on the Identity Registry contract
    *  AgentManager has to be set as agent on the Identity Registry smart contract to
process this function
    *  See {IIdentityRegistry-updateCountry}.
    *  Requires that `_onchainID` is set as WhiteListManager on the AgentManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callUpdateCountry(address _userAddress, uint16 _country, IIdentity
_onchainID) external {
```

```
        require(isWhiteListManager(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
WhiteList Manager");
        token.identityRegistry().updateCountry(_userAddress, _country);
    }

   /**
    *  @dev calls the `deleteIdentity` function on the Identity Registry contract
    *  AgentManager has to be set as agent on the Identity Registry smart contract to
process this function
    *  See {IIdentityRegistry-deleteIdentity}.
    *  Requires that `_onchainID` is set as WhiteListManager on the AgentManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callDeleteIdentity(address _userAddress, IIdentity _onchainID) external
{
        require(isWhiteListManager(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
WhiteList Manager");
        token.identityRegistry().deleteIdentity(_userAddress);
    }
}
```

AgentRole.sol

```
/**
 *     NOTICE
 *
 *     The T-REX software is licensed under a proprietary license or the GPL v.3.
 *     If you choose to receive it under the GPL v.3 license, the following applies:
 *     T-REX is a suite of smart contracts developed by Tokeny to manage and transfer
financial assets on the ethereum blockchain
 *
 *     Copyright (C) 2019, Tokeny sàrl.
 *
 *     This program is free software: you can redistribute it and/or modify
 *     it under the terms of the GNU General Public License as published by
 *     the Free Software Foundation, either version 3 of the License, or
 *     (at your option) any later version.
 *
 *     This program is distributed in the hope that it will be useful,
 *     but WITHOUT ANY WARRANTY; without even the implied warranty of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *     GNU General Public License for more details.
 *
 *     You should have received a copy of the GNU General Public License
 *     along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */

pragma solidity ^0.6.0;

import "./Roles.sol";
import "./Ownable.sol";
```

```
contract AgentRole is Ownable {
    using Roles for Roles.Role;

    event AgentAdded(address indexed _agent);
    event AgentRemoved(address indexed _agent);

    Roles.Role private _agents;

    modifier onlyAgent() {
        require(isAgent(msg.sender), "AgentRole: caller does not have the Agent
role");
        _;
    }

    function isAgent(address _agent) public view returns (bool) {
        return _agents.has(_agent);
    }

    function addAgent(address _agent) public onlyOwner {
        _agents.add(_agent);
        emit AgentAdded(_agent);
    }

    function removeAgent(address _agent) public onlyOwner {
        _agents.remove(_agent);
        emit AgentRemoved(_agent);
    }
}
```

AgentRoles.sol

```
/**
 *     NOTICE
 *
 *     The T-REX software is licensed under a proprietary license or the GPL v.3.
 *     If you choose to receive it under the GPL v.3 license, the following applies:
 *     T-REX is a suite of smart contracts developed by Tokeny to manage and transfer
financial assets on the ethereum blockchain
 *
 *     Copyright (C) 2019, Tokeny sàrl.
 *
 *     This program is free software: you can redistribute it and/or modify
 *     it under the terms of the GNU General Public License as published by
 *     the Free Software Foundation, either version 3 of the License, or
 *     (at your option) any later version.
 *
 *     This program is distributed in the hope that it will be useful,
 *     but WITHOUT ANY WARRANTY; without even the implied warranty of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *     GNU General Public License for more details.
 *
 *     You should have received a copy of the GNU General Public License
 *     along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */

pragma solidity ^0.6.0;
```

```
import "./Roles.sol";
import "./Ownable.sol";

contract AgentRoles is Ownable {
    using Roles for Roles.Role;

    event RoleAdded(address indexed _agent, string _role);
    event RoleRemoved(address indexed _agent, string _role);

    Roles.Role private _supplyModifiers;
    Roles.Role private _freezers;
    Roles.Role private _transferManagers;
    Roles.Role private _recoveryAgents;
    Roles.Role private _complianceAgents;
    Roles.Role private _whiteListManagers;
    Roles.Role private _agentAdmin;

    modifier onlyAdmin() {
        require(isOwner() || isAgentAdmin(_msgSender()), "Role: Sender is NOT
Admin");
        _;
    }

    /// AgentAdmin Role _agentAdmin

    function isAgentAdmin(address _agent) public view returns (bool) {
        return _agentAdmin.has(_agent);
    }

    function addAgentAdmin(address _agent) public onlyAdmin {
        _agentAdmin.add(_agent);
        string memory _role = "AgentAdmin";
        emit RoleAdded(_agent, _role);
    }

    function removeAgentAdmin(address _agent) public onlyAdmin {
        _agentAdmin.remove(_agent);
        string memory _role = "AgentAdmin";
        emit RoleRemoved(_agent, _role);
    }

    /// SupplyModifier Role _supplyModifiers

    function isSupplyModifier(address _agent) public view returns (bool) {
        return _supplyModifiers.has(_agent);
    }

    function addSupplyModifier(address _agent) public onlyAdmin {
        _supplyModifiers.add(_agent);
        string memory _role = "SupplyModifier";
        emit RoleAdded(_agent, _role);
    }

    function removeSupplyModifier(address _agent) public onlyAdmin {
        _supplyModifiers.remove(_agent);
        string memory _role = "SupplyModifier";
        emit RoleRemoved(_agent, _role);
    }

    /// Freezer Role _freezers
```

```
function isFreezer(address _agent) public view returns (bool) {
    return _freezers.has(_agent);
}

function addFreezer(address _agent) public onlyAdmin {
    _freezers.add(_agent);
    string memory _role = "Freezer";
    emit RoleAdded(_agent, _role);
}

function removeFreezer(address _agent) public onlyAdmin {
    _freezers.remove(_agent);
    string memory _role = "Freezer";
    emit RoleRemoved(_agent, _role);
}

/// TransferManager Role _transferManagers

function isTransferManager(address _agent) public view returns (bool) {
    return _transferManagers.has(_agent);
}

function addTransferManager(address _agent) public onlyAdmin {
    _transferManagers.add(_agent);
    string memory _role = "TransferManager";
    emit RoleAdded(_agent, _role);
}

function removeTransferManager(address _agent) public onlyAdmin {
    _transferManagers.remove(_agent);
    string memory _role = "TransferManager";
    emit RoleRemoved(_agent, _role);
}

/// RecoveryAgent Role _recoveryAgents

function isRecoveryAgent(address _agent) public view returns (bool) {
    return _recoveryAgents.has(_agent);
}

function addRecoveryAgent(address _agent) public onlyAdmin {
    _recoveryAgents.add(_agent);
    string memory _role = "RecoveryAgent";
    emit RoleAdded(_agent, _role);
}

function removeRecoveryAgent(address _agent) public onlyAdmin {
    _recoveryAgents.remove(_agent);
    string memory _role = "RecoveryAgent";
    emit RoleRemoved(_agent, _role);
}

/// ComplianceAgent Role _complianceAgents

function isComplianceAgent(address _agent) public view returns (bool) {
    return _complianceAgents.has(_agent);
}

function addComplianceAgent(address _agent) public onlyAdmin {
```

```
        _complianceAgents.add(_agent);
        string memory _role = "ComplianceAgent";
        emit RoleAdded(_agent, _role);
    }

    function removeComplianceAgent(address _agent) public onlyAdmin {
        _complianceAgents.remove(_agent);
        string memory _role = "ComplianceAgent";
        emit RoleRemoved(_agent, _role);
    }

    /// WhiteListManager Role _whiteListManagers

    function isWhiteListManager(address _agent) public view returns (bool) {
        return _whiteListManagers.has(_agent);
    }

    function addWhiteListManager(address _agent) public onlyAdmin {
        _whiteListManagers.add(_agent);
        string memory _role = "WhiteListManager";
        emit RoleAdded(_agent, _role);
    }

    function removeWhiteListManager(address _agent) public onlyAdmin {
        _whiteListManagers.remove(_agent);
        string memory _role = "WhiteListManager";
        emit RoleRemoved(_agent, _role);
    }
}
```

Ownable.sol

```
pragma solidity ^0.6.0;

import "openzeppelin-solidity/contracts/GSN/Context.sol";

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

    /**
     * @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(isOwner(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return _msgSender() == _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));
        _owner = address(0);
    }

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

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal virtual {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
```

kaspersky

OwnerManager.sol

```
/**
 *     NOTICE
 *
 *     The T-REX software is licensed under a proprietary license or the GPL v.3.
 *     If you choose to receive it under the GPL v.3 license, the following applies:
 *     T-REX is a suite of smart contracts developed by Tokeny to manage and transfer
financial assets on the ethereum blockchain
 *
 *     Copyright (C) 2019, Tokeny sàrl.
 *
 *     This program is free software: you can redistribute it and/or modify
 *     it under the terms of the GNU General Public License as published by
 *     the Free Software Foundation, either version 3 of the License, or
 *     (at your option) any later version.
 *
 *     This program is distributed in the hope that it will be useful,
 *     but WITHOUT ANY WARRANTY; without even the implied warranty of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *     GNU General Public License for more details.
 *
 *     You should have received a copy of the GNU General Public License
 *     along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */

pragma solidity ^0.6.0;

import "../token/IToken.sol";
import "../registry/IIdentityRegistry.sol";
import "../registry/ITrustedIssuersRegistry.sol";
import "../registry/IClaimTopicsRegistry.sol";
import "../compliance/ICompliance.sol";
import "./OwnerRoles.sol";
import "@onchain-id/solidity/contracts/IIdentity.sol";
import "@onchain-id/solidity/contracts/IClaimIssuer.sol";

contract OwnerManager is OwnerRoles {

    /// the token that is managed by this OwnerManager Contract
    IToken public token;

   /**
    *  @dev the constructor initiates the OwnerManager contract
    *  and sets msg.sender as owner of the contract
    *  @param _token the token managed by this OwnerManager contract
    */
    constructor (address _token) public {
        token = IToken(_token);
    }

   /**
    *  @dev calls the `setIdentityRegistry` function on the token contract
    *  OwnerManager has to be set as owner on the token smart contract to process
this function
    *  See {IToken-setIdentityRegistry}.
    *  Requires that `_onchainID` is set as RegistryAddressSetter on the OwnerManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
```

```
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callSetIdentityRegistry(address _identityRegistry, IIdentity _onchainID)
external {
        require(isRegistryAddressSetter(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Registry Address Setter");
        token.setIdentityRegistry(_identityRegistry);
    }

   /**
    *  @dev calls the `setCompliance` function on the token contract
    *  OwnerManager has to be set as owner on the token smart contract to process
this function
    *  See {IToken-setCompliance}.
    *  Requires that `_onchainID` is set as ComplianceSetter on the OwnerManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callSetCompliance(address _compliance, IIdentity _onchainID) external {
        require(isComplianceSetter(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Compliance Setter");
        token.setCompliance(_compliance);
    }

   /**
    *  @dev calls the `setName` function on the token contract
    *  OwnerManager has to be set as owner on the token smart contract to process
this function
    *  See {IToken-setName}.
    *  Requires that `_onchainID` is set as TokenInfoManager on the OwnerManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callSetTokenName(string calldata _name, IIdentity _onchainID) external {
        require(isTokenInfoManager(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Token Information Manager");
        token.setName(_name);
    }

   /**
    *  @dev calls the `setSymbol` function on the token contract
    *  OwnerManager has to be set as owner on the token smart contract to process
this function
    *  See {IToken-setSymbol}.
    *  Requires that `_onchainID` is set as TokenInfoManager on the OwnerManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callSetTokenSymbol(string calldata _symbol, IIdentity _onchainID)
external {
```

```
        require(isTokenInfoManager(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Token Information Manager");
        token.setSymbol(_symbol);
    }

   /**
    *  @dev calls the `setOnchainID` function on the token contract
    *  OwnerManager has to be set as owner on the token smart contract to process
this function
    *  See {IToken-setOnchainID}.
    *  Requires that `_tokenOnchainID` is set as TokenInfoManager on the OwnerManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callSetTokenOnchainID(address _tokenOnchainID, IIdentity _onchainID)
external {
        require(isTokenInfoManager(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Token Information Manager");
        token.setOnchainID(_tokenOnchainID);
    }

   /**
    *  @dev calls the `setClaimTopicsRegistry` function on the Identity Registry
contract
    *  OwnerManager has to be set as owner on the Identity Registry smart contract to
process this function
    *  See {IIdentityRegistry-setClaimTopicsRegistry}.
    *  Requires that `_onchainID` is set as RegistryAddressSetter on the OwnerManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callSetClaimTopicsRegistry(address _claimTopicsRegistry, IIdentity
_onchainID) external {
        require(isRegistryAddressSetter(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Registry Address Setter");
        token.identityRegistry().setClaimTopicsRegistry(_claimTopicsRegistry);
    }

   /**
    *  @dev calls the `setTrustedIssuersRegistry` function on the Identity Registry
contract
    *  OwnerManager has to be set as owner on the Identity Registry smart contract to
process this function
    *  See {IIdentityRegistry-setTrustedIssuersRegistry}.
    *  Requires that `_onchainID` is set as RegistryAddressSetter on the OwnerManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callSetTrustedIssuersRegistry(address _trustedIssuersRegistry, IIdentity
_onchainID) external {
```

```
        require(isRegistryAddressSetter(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
Registry Address Setter");
        token.identityRegistry().setTrustedIssuersRegistry(_trustedIssuersRegistry);
    }

   /**
    *  @dev calls the `addTrustedIssuer` function on the Trusted Issuers Registry
contract
    *  OwnerManager has to be set as owner on the Trusted Issuers Registry smart
contract to process this function
    *  See {ITrustedIssuersRegistry-addTrustedIssuer}.
    *  Requires that `_onchainID` is set as IssuersRegistryManager on the
OwnerManager contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callAddTrustedIssuer(IClaimIssuer _trustedIssuer, uint[] calldata
_claimTopics, IIdentity _onchainID) external {
        require(isIssuersRegistryManager(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
IssuersRegistryManager");
        token.identityRegistry().issuersRegistry().addTrustedIssuer(_trustedIssuer,
_claimTopics);
    }

   /**
    *  @dev calls the `removeTrustedIssuer` function on the Trusted Issuers Registry
contract
    *  OwnerManager has to be set as owner on the Trusted Issuers Registry smart
contract to process this function
    *  See {ITrustedIssuersRegistry-removeTrustedIssuer}.
    *  Requires that `_onchainID` is set as IssuersRegistryManager on the
OwnerManager contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callRemoveTrustedIssuer(IClaimIssuer _trustedIssuer, IIdentity
_onchainID) external {
        require(isIssuersRegistryManager(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
IssuersRegistryManager");

token.identityRegistry().issuersRegistry().removeTrustedIssuer(_trustedIssuer);
    }

   /**
    *  @dev calls the `updateIssuerClaimTopics` function on the Trusted Issuers
Registry contract
    *  OwnerManager has to be set as owner on the Trusted Issuers Registry smart
contract to process this function
    *  See {ITrustedIssuersRegistry-updateIssuerClaimTopics}.
    *  Requires that `_onchainID` is set as IssuersRegistryManager on the
OwnerManager contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
```

```
    function callUpdateIssuerClaimTopics(IClaimIssuer _trustedIssuer, uint[] calldata
_claimTopics, IIdentity _onchainID) external {
        require(isIssuersRegistryManager(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
IssuersRegistryManager");

token.identityRegistry().issuersRegistry().updateIssuerClaimTopics(_trustedIssuer,
_claimTopics);
    }

   /**
    *  @dev calls the `addClaimTopic` function on the Claim Topics Registry contract
    *  OwnerManager has to be set as owner on the Claim Topics Registry smart
contract to process this function
    *  See {IClaimTopicsRegistry-addClaimTopic}.
    *  Requires that `_onchainID` is set as ClaimRegistryManager on the OwnerManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callAddClaimTopic(uint256 _claimTopic, IIdentity _onchainID) external {
        require(isClaimRegistryManager(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
ClaimRegistryManager");
        token.identityRegistry().topicsRegistry().addClaimTopic(_claimTopic);
    }

   /**
    *  @dev calls the `removeClaimTopic` function on the Claim Topics Registry
contract
    *  OwnerManager has to be set as owner on the Claim Topics Registry smart
contract to process this function
    *  See {IClaimTopicsRegistry-removeClaimTopic}.
    *  Requires that `_onchainID` is set as ClaimRegistryManager on the OwnerManager
contract
    *  Requires that msg.sender is a MANAGEMENT KEY on `_onchainID`
    *  @param _onchainID the _onchainID contract of the caller, e.g. "i call this
function and i am Bob"
    */
    function callRemoveClaimTopic(uint256 _claimTopic, IIdentity _onchainID) external
{
        require(isClaimRegistryManager(address(_onchainID)) &&
_onchainID.keyHasPurpose(keccak256(abi.encode(msg.sender)), 1), "Role: Sender is NOT
ClaimRegistryManager");
        token.identityRegistry().topicsRegistry().removeClaimTopic(_claimTopic);
    }

   /**
    *  @dev calls the `transferOwnershipOnTokenContract` function on the token
contract
    *  OwnerManager has to be set as owner on the token smart contract to process
this function
    *  See {IToken-transferOwnershipOnTokenContract}.
    *  Requires that msg.sender is an Admin of the OwnerManager contract
    */
    function callTransferOwnershipOnTokenContract(address _newOwner) external
onlyAdmin {
        token.transferOwnershipOnTokenContract(_newOwner);
    }
```

```
   /**
    *  @dev calls the `transferOwnershipOnIdentityRegistryContract` function on the
Identity Registry contract
    *  OwnerManager has to be set as owner on the Identity Registry smart contract to
process this function
    *  See {IIdentityRegistry-transferOwnershipOnIdentityRegistryContract}.
    *  Requires that msg.sender is an Admin of the OwnerManager contract
    */
    function callTransferOwnershipOnIdentityRegistryContract(address _newOwner)
external onlyAdmin {

token.identityRegistry().transferOwnershipOnIdentityRegistryContract(_newOwner);
    }

   /**
    *  @dev calls the `transferOwnershipOnComplianceContract` function on the
Compliance contract
    *  OwnerManager has to be set as owner on the Compliance smart contract to
process this function
    *  See {ICompliance-transferOwnershipOnComplianceContract}.
    *  Requires that msg.sender is an Admin of the OwnerManager contract
    */
    function callTransferOwnershipOnComplianceContract(address _newOwner) external
onlyAdmin {
        token.compliance().transferOwnershipOnComplianceContract(_newOwner);
    }

   /**
    *  @dev calls the `transferOwnershipOnClaimTopicsRegistryContract` function on
the Claim Topics Registry contract
    *  OwnerManager has to be set as owner on the Claim Topics registry smart
contract to process this function
    *  See {IClaimTopicsRegistry-transferOwnershipOnClaimTopicsRegistryContract}.
    *  Requires that msg.sender is an Admin of the OwnerManager contract
    */
    function callTransferOwnershipOnClaimTopicsRegistryContract(address _newOwner)
external onlyAdmin {

token.identityRegistry().topicsRegistry().transferOwnershipOnClaimTopicsRegistryContr
act(_newOwner);
    }

   /**
    *  @dev calls the `transferOwnershipOnIssuersRegistryContract` function on the
Trusted Issuers Registry contract
    *  OwnerManager has to be set as owner on the Trusted Issuers registry smart
contract to process this function
    *  See {ITrustedIssuersRegistry-transferOwnershipOnIssuersRegistryContract}.
    *  Requires that msg.sender is an Admin of the OwnerManager contract
    */
    function callTransferOwnershipOnIssuersRegistryContract(address _newOwner)
external onlyAdmin {

token.identityRegistry().issuersRegistry().transferOwnershipOnIssuersRegistryContract
(_newOwner);
    }

   /**
    *  @dev calls the `addAgentOnTokenContract` function on the token contract
```

```
    *  OwnerManager has to be set as owner on the token smart contract to process
this function
    *  See {IToken-addAgentOnTokenContract}.
    *  Requires that msg.sender is an Admin of the OwnerManager contract
    */
    function callAddAgentOnTokenContract(address _agent) external onlyAdmin {
        token.addAgentOnTokenContract(_agent);
    }

   /**
    *  @dev calls the `removeAgentOnTokenContract` function on the token contract
    *  OwnerManager has to be set as owner on the token smart contract to process
this function
    *  See {IToken-removeAgentOnTokenContract}.
    *  Requires that msg.sender is an Admin of the OwnerManager contract
    */
    function callRemoveAgentOnTokenContract(address _agent) external onlyAdmin {
        token.removeAgentOnTokenContract(_agent);
    }

   /**
    *  @dev calls the `addAgentOnIdentityRegistryContract` function on the Identity
Registry contract
    *  OwnerManager has to be set as owner on the Identity Registry smart contract to
process this function
    *  See {IIdentityRegistry-addAgentOnIdentityRegistryContract}.
    *  Requires that msg.sender is an Admin of the OwnerManager contract
    */
    function callAddAgentOnIdentityRegistryContract(address _agent) external
onlyAdmin {
        token.identityRegistry().addAgentOnIdentityRegistryContract(_agent);
    }

   /**
    *  @dev calls the `removeAgentOnIdentityRegistryContract` function on the
Identity Registry contract
    *  OwnerManager has to be set as owner on the Identity Registry smart contract to
process this function
    *  See {IIdentityRegistry-removeAgentOnIdentityRegistryContract}.
    *  Requires that msg.sender is an Admin of the OwnerManager contract
    */
    function callRemoveAgentOnIdentityRegistryContract(address _agent) external
onlyAdmin {
        token.identityRegistry().removeAgentOnIdentityRegistryContract(_agent);
    }
}
```

OwnerRoles.sol

```
/**
 *     NOTICE
 *
 *     The T-REX software is licensed under a proprietary license or the GPL v.3.
 *     If you choose to receive it under the GPL v.3 license, the following applies:
 *     T-REX is a suite of smart contracts developed by Tokeny to manage and transfer
financial assets on the ethereum blockchain
 *
```

```
pragma solidity ^0.6.0;

import "./Roles.sol";
import "./Ownable.sol";

contract OwnerRoles is Ownable {
    using Roles for Roles.Role;

    event RoleAdded(address indexed _owner, string _role);
    event RoleRemoved(address indexed _owner, string _role);

    Roles.Role private _ownerAdmin;
    Roles.Role private _registryAddressSetter;
    Roles.Role private _complianceSetter;
    Roles.Role private _claimRegistryManager;
    Roles.Role private _issuersRegistryManager;
    Roles.Role private _tokenInfoManager;

    modifier onlyAdmin() {
        require(isOwner() || isOwnerAdmin(_msgSender()), "Role: Sender is NOT
Admin");
        _;
    }

    /// OwnerAdmin Role _ownerAdmin

    function isOwnerAdmin(address _owner) public view returns (bool) {
        return _ownerAdmin.has(_owner);
    }

    function addOwnerAdmin(address _owner) public onlyAdmin {
        _ownerAdmin.add(_owner);
        string memory _role = "OwnerAdmin";
        emit RoleAdded(_owner, _role);
    }

    function removeOwnerAdmin(address _owner) public onlyAdmin {
        _ownerAdmin.remove(_owner);
        string memory _role = "OwnerAdmin";
        emit RoleRemoved(_owner, _role);
    }

    /// RegistryAddressSetter Role _registryAddressSetter
```

```
function isRegistryAddressSetter(address _owner) public view returns (bool) {
    return _registryAddressSetter.has(_owner);
}

function addRegistryAddressSetter(address _owner) public onlyAdmin {
    _registryAddressSetter.add(_owner);
    string memory _role = "RegistryAddressSetter";
    emit RoleAdded(_owner, _role);
}

function removeRegistryAddressSetter(address _owner) public onlyAdmin {
    _registryAddressSetter.remove(_owner);
    string memory _role = "RegistryAddressSetter";
    emit RoleRemoved(_owner, _role);
}

/// ComplianceSetter Role _complianceSetter

function isComplianceSetter(address _owner) public view returns (bool) {
    return _complianceSetter.has(_owner);
}

function addComplianceSetter(address _owner) public onlyAdmin {
    _complianceSetter.add(_owner);
    string memory _role = "ComplianceSetter";
    emit RoleAdded(_owner, _role);
}

function removeComplianceSetter(address _owner) public onlyAdmin {
    _complianceSetter.remove(_owner);
    string memory _role = "ComplianceSetter";
    emit RoleRemoved(_owner, _role);
}

/// ClaimRegistryManager Role _claimRegistryManager

function isClaimRegistryManager(address _owner) public view returns (bool) {
    return _claimRegistryManager.has(_owner);
}

function addClaimRegistryManager(address _owner) public onlyAdmin {
    _claimRegistryManager.add(_owner);
    string memory _role = "ClaimRegistryManager";
    emit RoleAdded(_owner, _role);
}

function removeClaimRegistryManager(address _owner) public onlyAdmin {
    _claimRegistryManager.remove(_owner);
    string memory _role = "ClaimRegistryManager";
    emit RoleRemoved(_owner, _role);
}

/// IssuersRegistryManager Role _issuersRegistryManager

function isIssuersRegistryManager(address _owner) public view returns (bool) {
    return _issuersRegistryManager.has(_owner);
}

function addIssuersRegistryManager(address _owner) public onlyAdmin {
    _issuersRegistryManager.add(_owner);
```

```
        string memory _role = "IssuersRegistryManager";
        emit RoleAdded(_owner, _role);
    }

    function removeIssuersRegistryManager(address _owner) public onlyAdmin {
        _issuersRegistryManager.remove(_owner);
        string memory _role = "IssuersRegistryManager";
        emit RoleRemoved(_owner, _role);
    }

    /// TokenInfoManager Role _tokenInfoManager

    function isTokenInfoManager(address _owner) public view returns (bool) {
        return _tokenInfoManager.has(_owner);
    }

    function addTokenInfoManager(address _owner) public onlyAdmin {
        _tokenInfoManager.add(_owner);
        string memory _role = "TokenInfoManager";
        emit RoleAdded(_owner, _role);
    }

    function removeTokenInfoManager(address _owner) public onlyAdmin {
        _tokenInfoManager.remove(_owner);
        string memory _role = "TokenInfoManager";
        emit RoleRemoved(_owner, _role);
    }
}
```

Roles.sol

```
pragma solidity ^0.6.0;

/**
 * @title Roles
 * @dev Library for managing addresses assigned to a Role.
 */
library Roles {
    struct Role {
        mapping (address => bool) bearer;
    }

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

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

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

kaspersky

www.kaspersky.com/
www.securelist.com