README.md

**What is zkMe Network?**

The web3 credential network

zkMe is a decentralized Web3 credential network that leverages the power of zero-knowledge proofs to enable secure and private credential issuance and verification.

With zkMe, users can disclose their credentials to authorized parties selectively, without compromising their privacy, and enjoy greater control over their digital identities.

**As an individual**

You, privately verified.

* zkMe verifies user credentials without disclosing any personal information to anyone!
* With zkMe, you can be confident that all your personal information is protected and your interaction with Web3 protocols are secure.

**As a Web3 protocol**

You can build possibilities based on zk-credentials.

* Eliminate bots and fake accounts instantly.
* Comply with upcoming regulations by performing zkKYC.
* Verify users' identity-based data and unlock new possibilities.

solidity/contracts/ZKMESBTUpgradeable.sol

// SPDX-License-Identifier: MIT

pragma solidity **^**0.8.17**;**

**import** "./KycData/KYCDataLib.sol"**;**

**import** "./KycData/IKYCDataReadable.sol"**;**

**import** "./interfaces/IERC721MetadataUpgradeable.sol"**;**

**import** "@openzeppelin/contracts-upgradeable/interfaces/IERC165Upgradeable.sol"**;**

**import** "@openzeppelin/contracts-upgradeable/interfaces/IERC721Upgradeable.sol"**;**

**import** "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol"**;**

**import** "@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol"**;**

**import** "@openzeppelin/contracts-upgradeable/utils/CountersUpgradeable.sol"**;**

**import** "@openzeppelin/contracts-upgradeable/utils/structs/EnumerableMapUpgradeable.sol"**;**

**import** "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol"**;**

/\*\*

\* An experiment in zkMe Soul Bound Tokens (ZKMESBT's)

\*/

contract ZKMESBTUpgradeable is

Initializable**,**

AccessControlUpgradeable**,**

IKYCDataReadable**,**

IERC721MetadataUpgradeable // 本质上是一个ERC721 NFT token，一个id对应一个NFT

**{**

using StringsUpgradeable **for** uint256**;**

using CountersUpgradeable **for** CountersUpgradeable**.**Counter**;**

using EnumerableMapUpgradeable **for** EnumerableMapUpgradeable**.**AddressToUintMap**;**

using EnumerableMapUpgradeable **for** EnumerableMapUpgradeable**.**UintToAddressMap**;**

EnumerableMapUpgradeable**.**UintToAddressMap **private** \_ownerMap**;**

EnumerableMapUpgradeable**.**AddressToUintMap **private** \_tokenMap**;**

CountersUpgradeable**.**Counter **private** \_tokenId**;**

string **public** name**; // ERC721 name**

string **public** **symbol; // ERC721 symbol**

string **private** \_baseTokenURI**;**

mapping**(**uint **=>** KYCDataLib**.**UserData**)** **private** \_kycMap**;**

bytes32 **public** constant OPERATOR\_ROLE **=** keccak256**(**"OPERATOR\_ROLE"**);**

/\*\*

\* @dev Initializes the contract by setting a `name` and a `symbol` to the token collection.

\*/

**function** initialize**(**

string memory name\_**,**

string memory symbol\_**,**

address admin\_ // 写死一个zkme的管理员地址

**)** **public** reinitializer**(**1**)** **{**

name **=** name\_**;**

**symbol** **=** symbol\_**;**

\_grantRole**(**DEFAULT\_ADMIN\_ROLE**,** admin\_**);**

\_grantRole**(**OPERATOR\_ROLE**,** admin\_**);**

**}**

**function** attest**(**

address to

**)** external onlyRole**(**OPERATOR\_ROLE**)** returns **(**uint256**)** **{**

**require(**to **!=** address**(**0**),** "Empty address is not allowed"**);**

**require(!**\_tokenMap**.**contains**(**to**),** "zkMeSBT already exists"**);**

\_tokenId**.**increment**();**

uint256 tokenId **=** \_tokenId**.**current**();**

\_tokenMap**.set(**to**,** tokenId**);**

\_ownerMap**.set(**tokenId**,** to**);**

emit Attest**(**to**,** tokenId**);**

emit Transfer**(**address**(**0**),** to**,** tokenId**);**

**return** tokenId**;**

**}**

**function** revoke**(**

address **from,**

uint256 tokenId

**)** external onlyRole**(**OPERATOR\_ROLE**)** **{**

**require(from** **!=** address**(**0**),** "Empty address is not allowed"**);**

**require(**

\_tokenMap**.**contains**(from),**

"The account does not have the zkMeSBT"

**);**

\_tokenMap**.**remove**(from);**

\_ownerMap**.**remove**(**tokenId**);**

emit Revoke**(from,** tokenId**);**

emit Transfer**(from,** address**(**0**),** tokenId**);**

**}**

**function** burn**(**uint256 tokenId**)** external **{**

address sender **=** \_msgSender**();**

**require(**

\_tokenMap**.**contains**(**sender**),**

"The account does not have the zkMeSBT"

**);**

\_tokenMap**.**remove**(**sender**);**

\_ownerMap**.**remove**(**tokenId**);**

emit Burn**(**sender**,** tokenId**);**

emit Transfer**(**sender**,** address**(**0**),** tokenId**);**

**}**

**function** getKycData**(**

uint256 tokenId

**)** **public** view returns **(**KYCDataLib**.**UserData memory**)** **{**

**require(**\_ownerMap**.**contains**(**tokenId**),** "The zkMeSBT does not exist"**);**

**return** \_kycMap**[**tokenId**];**

**}**

**function** setKycData**(**

uint256 tokenId**,**

string calldata key**,**

uint256 validity**,**

string calldata data**,**

string**[]** calldata questions

**)** **public** onlyRole**(**OPERATOR\_ROLE**)** **{**

**require(**\_ownerMap**.**contains**(**tokenId**),** "The zkMeSBT does not exist"**);**

**require(**

validity **>** block**.**timestamp**,**

"The expiration date is too closed"

**);**

**if** **(**bytes**(**\_kycMap**[**tokenId**].**key**).**length **!=** 0**)** **{**

**require(**

keccak256**(**bytes**(**\_kycMap**[**tokenId**].**key**))** **==** keccak256**(**bytes**(**key**)),**

"Dismatched user key"

**);**

**}**

\_setKycData**(**tokenId**,** key**,** validity**,** data**,** questions**);**

**}**

**function** \_setKycData**(**

uint256 tokenId**,**

string calldata key**,**

uint256 validity**,**

string calldata data**,**

string**[]** calldata questions

**)** internal **{**

\_kycMap**[**tokenId**]** **=** KYCDataLib**.**UserData**(**key**,** validity**,** data**,** questions**);**

**}**

/\*\*

\* @dev Update \_baseTokenURI

\*/

**function** setBaseTokenURI**(**

string calldata uri

**)** **public** onlyRole**(**DEFAULT\_ADMIN\_ROLE**)** **{**

\_baseTokenURI **=** uri**;**

**}**

**function** balanceOf**(**

address owner

**)** external view override**(**IZKMESBT721Upgradeable**)** returns **(**uint256**)** **{**

**(**bool success**,** **)** **=** \_tokenMap**.**tryGet**(**owner**);**

**return** success **?** 1 **:** 0**;**

**}**

**function** tokenIdOf**(**address **from)** external view returns **(**uint256**)** **{**

**return** \_tokenMap**.get(from,** "The address does not have any zkMeSBT"**);**

**}**

**function** ownerOf**(**

uint256 tokenId

**)** external view override**(**IZKMESBT721Upgradeable**)** returns **(**address**)** **{**

**return** \_ownerMap**.get(**tokenId**,** "Invalid tokenId"**);**

**}**

**function** totalSupply**()** external view override returns **(**uint256**)** **{**

**return** \_tokenId**.**current**();**

**}**

**function** isOperator**(**address account**)** external view returns **(**bool**)** **{**

**return** hasRole**(**OPERATOR\_ROLE**,** account**);**

**}**

**function** isAdmin**(**address account**)** external view returns **(**bool**)** **{**

**return** hasRole**(**DEFAULT\_ADMIN\_ROLE**,** account**);**

**}**

/\*\*

\* @dev See {IERC721Metadata-tokenURI}.

\*

\*/

**function** tokenURI**(**uint256 tokenId**)** external view returns **(**string memory**)** **{**

**return**

bytes**(**\_baseTokenURI**).**length **>** 0

**?** string**(**abi**.**encodePacked**(**\_baseTokenURI**,** tokenId**.**toString**()))**

**:** ""**;**

**}**

/\*\*

\* @dev See {IERC165-supportsInterface}.

\*/

**function** supportsInterface**(**

bytes4 interfaceId

**)**

**public**

view

virtual

override**(**AccessControlUpgradeable**,** IERC165Upgradeable**)**

returns **(**bool**)**

**{**

**return**

interfaceId **==** **type(**IERC721Upgradeable**).**interfaceId **||**

interfaceId **==** **type(**IERC721MetadataUpgradeable**).**interfaceId **||**

**super.**supportsInterface**(**interfaceId**);**

**}**

**}**

solidity/contracts/KycData/KYCDataLib.sol

// SPDX-License-Identifier: MIT

pragma solidity **^**0.8.17**;**

library KYCDataLib **{**

// keccak256("EXPIRATION\_ERROR")

bytes32 **private** constant EXPIRATION\_ERROR **=**

0xfd53a8e7532291172be9639256053abc4567e8ac9c856d4cce12b1024ab10967**;**

// keccak256("VERIFIED")

bytes32 **private** constant VERIFIED **=**

0x3b9099870b8ae4badd49e59f30fc613f918d145d89048031bb3fca631cef16cb**;**

struct UserData **{**

string key**;**

uint256 validity**;**

string data**;**

string**[]** questions**;**

**}**

**}**

solidity/contracts/ZKMEConfUpgradeable.sol

// SPDX-License-Identifier: MIT

pragma solidity **^**0.8.17**;**

**import** "./interfaces/IZKMEConfUpgradeable.sol"**;**

**import** "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol"**;**

**import** "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol"**;**

contract ZKMEConfUpgradeable is

Initializable**,**

AccessControlUpgradeable**,**

IZKMEConfUpgradeable

**{**

mapping**(**address **=>** string**[])** **private** \_questionMap**;**

bytes32 **public** constant OPERATOR\_ROLE **=** keccak256**(**"OPERATOR\_ROLE"**);**

uint256 **public** constant OPERATOR\_GRANT **=** 0**;**

**function** initialize**(**address admin\_**)** **public** reinitializer**(**1**)** **{**

\_grantRole**(**DEFAULT\_ADMIN\_ROLE**,** admin\_**);**

\_grantRole**(**OPERATOR\_ROLE**,** admin\_**);**

**}**

**function** grantOperator**(**address operator**)** external onlyRole**(**OPERATOR\_ROLE**)** **{**

\_grantRole**(**OPERATOR\_ROLE**,** operator**);**

emit Grant**(**operator**,** OPERATOR\_GRANT**);**

**}**

**function** isOperator**(**address account**)** **public** view returns **(**bool**)** **{**

**return** hasRole**(**OPERATOR\_ROLE**,** account**);**

**}**

**function** setQuestions**(**

address cooperator**,**

string**[]** memory questions

**)** external onlyRole**(**OPERATOR\_ROLE**)** **{**

**require(**cooperator **!=** address**(**0**),** "empty address is not allowed"**);**

\_questionMap**[**cooperator**]** **=** questions**;**

emit SetQuestion**(**cooperator**);**

**}**

**function** getQuestions**(**

address cooperator

**)** external view returns **(**string**[]** memory**)** **{**

**require(**cooperator **!=** address**(**0**),** "empty address is not allowed"**);**

**return** \_questionMap**[**cooperator**];**

**}**

**}**

solidity/contracts/ZKMEVerifyUpgradeable.sol

// SPDX-License-Identifier: MIT

pragma solidity **^**0.8.17**;**

**import** "./KycData/KYCDataLib.sol"**;**

**import** "./KycData/IKYCDataReadable.sol"**;**

**import** "./interfaces/IZKMESBT721Upgradeable.sol"**;**

**import** "./interfaces/IZKMEVerifyUpgradeable.sol"**;**

**import** "./interfaces/IZKMEApprove.sol"**;**

**import** "./interfaces/IZKMEConfUpgradeable.sol"**;**

**import** "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol"**;**

**import** "@openzeppelin/contracts-upgradeable/utils/structs/EnumerableSetUpgradeable.sol"**;**

**import** "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol"**;**

contract ZKMEVerifyUpgradeable is

Initializable**,**

AccessControlUpgradeable**,**

IZKMEVerifyUpgradeable**,**

IZKMEApprove

**{**

using EnumerableSetUpgradeable **for** EnumerableSetUpgradeable**.**UintSet**;**

mapping**(**address **=>** mapping**(**address **=>** uint256**))** **private** \_puMap**;**

mapping**(**address **=>** mapping**(**uint256 **=>** KYCDataLib**.**UserData**))**

**private** \_kycDataMap**;**

mapping**(**address **=>** EnumerableSetUpgradeable**.**UintSet**)** **private** \_approveMap**;**

address **private** \_sbt\_contract**;**

address **private** \_conf\_contract**;**

bytes32 **public** constant OPERATOR\_ROLE **=** keccak256**(**"OPERATOR\_ROLE"**);**

bytes32 **public** constant COOPERATOR\_ROLE **=** keccak256**(**"COOPERATOR\_ROLE"**);**

bytes32 **public** constant INSPECTOR\_ROLE **=** keccak256**(**"INSPECTOR\_ROLE"**);**

uint256 **public** constant OPERATOR\_GRANT **=** 0**;**

uint256 **public** constant COOPERATOR\_GRANT **=** 1**;**

uint256 **public** constant INSPECTOR\_GRANT **=** 2**;**

**function** initialize**(**

address admin\_**,**

address sbt\_contract\_**,**

address conf\_contract\_

**)** **public** reinitializer**(**1**)** **{**

\_sbt\_contract **=** sbt\_contract\_**;**

\_conf\_contract **=** conf\_contract\_**;**

\_grantRole**(**DEFAULT\_ADMIN\_ROLE**,** admin\_**);**

\_grantRole**(**OPERATOR\_ROLE**,** admin\_**);**

\_grantRole**(**INSPECTOR\_ROLE**,** admin\_**);**

**}**

**function** updateSbtContract**(**

address contract\_

**)** external onlyRole**(**OPERATOR\_ROLE**)** **{**

\_sbt\_contract **=** contract\_**;**

**}**

**function** updateConfContract**(**

address contract\_

**)** external onlyRole**(**OPERATOR\_ROLE**)** **{**

\_conf\_contract **=** contract\_**;**

**}**

**function** grantOperator**(**

address operator

**)** external onlyRole**(**DEFAULT\_ADMIN\_ROLE**)** **{**

\_grantRole**(**OPERATOR\_ROLE**,** operator**);**

emit Grant**(**operator**,** OPERATOR\_GRANT**);**

**}**

**function** grantCooperator**(**

address cooperator

**)** external onlyRole**(**OPERATOR\_ROLE**)** **{**

\_grantRole**(**COOPERATOR\_ROLE**,** cooperator**);**

emit Grant**(**cooperator**,** COOPERATOR\_GRANT**);**

**}**

**function** grantInspector**(**

address inspector

**)** external onlyRole**(**OPERATOR\_ROLE**)** **{**

\_grantRole**(**INSPECTOR\_ROLE**,** inspector**);**

emit Grant**(**inspector**,** INSPECTOR\_GRANT**);**

**}**

**function** isOperator**(**address account**)** **public** view returns **(**bool**)** **{**

**return** hasRole**(**OPERATOR\_ROLE**,** account**);**

**}**

**function** isCooperator**(**address account**)** **public** view returns **(**bool**)** **{**

**return** hasRole**(**COOPERATOR\_ROLE**,** account**);**

**}**

**function** isInspector**(**address account**)** external view returns **(**bool**)** **{**

**return** hasRole**(**INSPECTOR\_ROLE**,** account**);**

**}**

**function** approve**(**

address cooperator**,**

uint256 tokenId**,**

string memory cooperatorThresholdKey

**)** external **{**

**require(**isCooperator**(**cooperator**),** "Invalid cooperator address."**);**

address tokenOwner **=** IZKMESBT721Upgradeable**(**\_sbt\_contract**).**ownerOf**(**

tokenId

**);**

**require(**

tokenOwner **==** \_msgSender**()** **||** isOperator**(**\_msgSender**()),**

"The invoker does not have the zkMeSBT."

**);**

KYCDataLib**.**UserData memory userData **=** IKYCDataReadable**(**\_sbt\_contract**)**

**.**getKycData**(**tokenId**);**

userData**.**key **=** cooperatorThresholdKey**;**

\_kycDataMap**[**cooperator**][**tokenId**]** **=** userData**;**

\_approveMap**[**cooperator**].**add**(**tokenId**);**

\_puMap**[**cooperator**][**tokenOwner**]** **=** tokenId**;**

emit Approve**(**cooperator**,** tokenId**);**

**}**

**function** revoke**(**address cooperator**,** uint256 tokenId**)** external **{**

**require(**

IZKMESBT721Upgradeable**(**\_sbt\_contract**).**ownerOf**(**tokenId**)** **==**

\_msgSender**()** **||**

isOperator**(**\_msgSender**()),**

"The invoker does not have the sbt"

**);**

\_approveMap**[**cooperator**].**remove**(**tokenId**);**

emit Revoke**(**cooperator**,** tokenId**);**

**}**

**function** \_matching**(**

string**[]** memory project**,**

string**[]** memory user

**)** **private** pure returns **(**bool**)** **{**

bool found **=** **false;**

**for** **(**uint i **=** 0**;** i **<** project**.**length**;** i**++)** **{**

**for** **(**uint j **=** 0**;** j **<** user**.**length**;** j**++)** **{**

**if** **(**keccak256**(**bytes**(**project**[**i**]))** **==** keccak256**(**bytes**(**user**[**j**])))** **{**

found **=** **true;**

**}**

**}**

**if** **(**found**)** **{**

found **=** **false;**

**}** **else** **{**

**return** **false;**

**}**

**}**

**return** **true;**

**}**

**function** verify**(**

address cooperator**,**

address user

**)** **public** view returns **(**bool**)** **{**

uint256 tokenId **=** IZKMESBT721Upgradeable**(**\_sbt\_contract**).**tokenIdOf**(**user**);**

KYCDataLib**.**UserData memory userData **=** IKYCDataReadable**(**\_sbt\_contract**)**

**.**getKycData**(**tokenId**);**

**if** **(**userData**.**validity **<** block**.**timestamp**)** **{**

**return** **false;**

**}**

string**[]** memory project **=** IZKMEConfUpgradeable**(**\_conf\_contract**)**

**.**getQuestions**(**cooperator**);**

**if** **(**project**.**length **==** 0**)** **{**

**return** **false;**

**}**

**return** \_matching**(**project**,** userData**.**questions**);**

**}**

**function** hasApproved**(**

address cooperator**,**

address user

**)** **public** view returns **(**bool**)** **{**

uint256 tokenId **=** \_getUserTokenId**(**cooperator**,** user**);**

**return** tokenId **!=** 0 **&&** \_approveMap**[**cooperator**].**contains**(**tokenId**);**

**}**

**function** getUserTokenId**(**

address user

**)** external view onlyRole**(**COOPERATOR\_ROLE**)** returns **(**uint256**)** **{**

**return**

hasApproved**(**\_msgSender**(),** user**)**

**?** \_getUserTokenId**(**\_msgSender**(),** user**)**

**:** 0**;**

**}**

**function** getUserTokenIdForOperator**(**

address cooperator**,**

address user

**)** external view onlyRole**(**OPERATOR\_ROLE**)** returns **(**uint256**)** **{**

**return** \_getUserTokenId**(**cooperator**,** user**);**

**}**

**function** getUserData**(**

address user

**)** external view returns **(**KYCDataLib**.**UserData memory**)** **{**

uint256 tokenId **=** \_getUserTokenId**(**\_msgSender**(),** user**);**

**require(**

hasApproved**(**\_msgSender**(),** user**),**

"The user didn't approve the zkMeSBT."

**);**

**return** \_getUserKycData**(**\_msgSender**(),** tokenId**);**

**}**

**function** getUserDataForOperator**(**

address cooperator**,**

address user

**)**

external

view

onlyRole**(**OPERATOR\_ROLE**)**

returns **(**KYCDataLib**.**UserData memory**)**

**{**

uint256 tokenId **=** \_getUserTokenId**(**cooperator**,** user**);**

**require(**tokenId **!=** 0**,** "The user didn't approve the zkMeSBT."**);**

**return** \_getUserKycData**(**cooperator**,** tokenId**);**

**}**

**function** getUserDataForInspector**(**

address party**,**

address user

**)**

**public**

view

onlyRole**(**INSPECTOR\_ROLE**)**

returns **(**KYCDataLib**.**UserData memory**)**

**{**

uint256 tokenId **=** \_getUserTokenId**(**party**,** user**);**

**require(**tokenId **!=** 0**,** "The user didn't approve the zkMeSBT."**);**

**return** \_getUserKycData**(**party**,** tokenId**);**

**}**

**function** getApprovedTokenId**(**

uint256 start**,**

uint256 pageSize

**)** **public** view onlyRole**(**COOPERATOR\_ROLE**)** returns **(**uint256**[**50**]** memory**)** **{**

**return** \_getApprovedTokenIdList**(**\_msgSender**(),** start**,** pageSize**);**

**}**

**function** getApprovedTokenIdForOperator**(**

address cooperator**,**

uint256 start**,**

uint256 pageSize

**)** **public** view onlyRole**(**OPERATOR\_ROLE**)** returns **(**uint256**[**50**]** memory**)** **{**

**return** \_getApprovedTokenIdList**(**cooperator**,** start**,** pageSize**);**

**}**

**function** getApprovedLength**()**

**public**

view

onlyRole**(**COOPERATOR\_ROLE**)**

returns **(**uint256**)**

**{**

**return** \_getApprovedLength**(**\_msgSender**());**

**}**

**function** getApprovedLengthForOperator**(**

address cooperator

**)** **public** view onlyRole**(**OPERATOR\_ROLE**)** returns **(**uint256**)** **{**

**return** \_getApprovedLength**(**cooperator**);**

**}**

**function** \_getUserKycData**(**

address cooperator**,**

uint256 tokenId

**)** **private** view returns **(**KYCDataLib**.**UserData memory**)** **{**

**return** \_kycDataMap**[**cooperator**][**tokenId**];**

**}**

**function** \_getApprovedLength**(**

address cooperator

**)** **private** view returns **(**uint256**)** **{**

**return** \_approveMap**[**cooperator**].**length**();**

**}**

**function** \_getUserTokenId**(**

address cooperator**,**

address user

**)** **private** view returns **(**uint256**)** **{**

**return** \_puMap**[**cooperator**][**user**];**

**}**

**function** \_getApprovedTokenIdList**(**

address cooperator**,**

uint256 start**,**

uint256 pageSize

**)** **private** view returns **(**uint256**[**50**]** memory**)** **{**

**require(**pageSize **<=** 50**,** "Page size must be less than 50."**);**

uint256**[**50**]** memory tokenIdList**;**

uint256 totalLength **=** \_approveMap**[**cooperator**].**length**();**

uint256 end **=** start **+** pageSize **>=** totalLength

**?** totalLength

**:** start **+** pageSize**;**

**for** **(**uint256 i **=** start**;** i **<** end**;** i**++)** **{**

tokenIdList**[**i **-** start**]** **=** \_approveMap**[**cooperator**].**at**(**i**);**

**}**

**return** tokenIdList**;**

**}**

**}**