Write a Solidity Smart Contract for a Non-Fungible  
Token (NFT) which can able to perform Identity  
Verification and Authentication as they can contain  
unique information about an individual that is  
Cryptographically secure.

Ans: // SPDX-License-Identifier: MIT

pragma solidity >0.8.0;

import"@openzeppelin/contracts/token/ERC721/ERC721.sol";

import"@openzeppelin/contracts/access/Ownable.sol";

import"@openzeppelin/contracts/utils/Counters.sol";

contract NFTIdentityToken is ERC721, Ownable{

    Counters.Counter private \_tokenIdCounter;

    using Counters for Counters.Counter;

    mapping (address=>bool) \_verified;

    constructor()ERC721 ("NFT Identity Token","NFTIT") {

    }

    function verify(address \_address) external onlyOwner {

        \_verified[\_address]=true;

    }

    function revoke(address \_address) external onlyOwner {

        \_verified[\_address]=false;

    }

    function checkVerifiiedOrNot(address \_addr) external view returns (bool) {

        return \_verified[\_addr];

    }

    function safeMint(address to) public onlyOwner{

        \_tokenIdCounter.increment();

        uint256 tokenId= \_tokenIdCounter.current();

        require(!\_exist(tokenId), "token already exists");

        \_safeMint(to,tokenId);

    }

    function \_beforeTokenTransfer(address from, address to,uint256 tokenId,uint256 batchSize) internal override (ERC721) {

        super.\_beforeTokenTransfer(from, to, tokenId, batchSize);

        if(from!=address(0)&&to!=address(0)){

            require(\_verified[from],"'from' is not a verified address");

            require(\_verified[to],"'to' is not a verified address");

        }

        else if(from ==address(0)&&to != address(0)){

            require(\_verified[to],"'to' is not a verified  address");

        }

    }

}