```
//ERC721OTP save-gas (one time passward card NFT)
//2020-07-05
//Author : Katsuya Nishizawa
//NZRI

//openzeppelin v3.1.0 (openzeppelin v3.1.0  is MIT License)
//based ERC721-openzeppelin v3.1.0

pragma solidity ^0.6.0;

import "../../GSN/Context.sol";
import "./IERC721.sol";
import "./IERC721Metadata.sol";
import "./IERC721Enumerable.sol";
import "./IERC721Receiver.sol";
import "../../introspection/ERC165.sol";
import "../../math/SafeMath.sol";
import "../../utils/Address.sol";
import "../../utils/EnumerableSet.sol";
import "../../utils/EnumerableMap.sol";
import "../../utils/Strings.sol";

/**
 * @title ERC721 Non-Fungible Token Standard basic implementation
 * @dev see https://eips.ethereum.org/EIPS/eip-721
 */
contract ERC721OTPC is Context, ERC165, IERC721, IERC721Metadata,
IERC721Enumerable {
    using SafeMath for uint256;
    using Address for address;
    using EnumerableSet for EnumerableSet.UintSet;
    using EnumerableMap for EnumerableMap.UintToAddressMap;
    using Strings for uint256;

    // Equals to
`bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`
    // which can be also obtained as
`IERC721Receiver(0).onERC721Received.selector`
    bytes4 private constant _ERC721_RECEIVED = 0x150b7a02;
    // Mapping from holder address to their (enumerable) set of owned tokens
    mapping (address => EnumerableSet.UintSet) private _holderTokens;

    // Enumerable mapping from token ids to their owners
    EnumerableMap.UintToAddressMap private _tokenOwners;

    // Mapping from token ID to approved address
    mapping (uint256 => address) private _tokenApprovals;

    // Mapping from owner to operator approvals
    mapping (address => mapping (address => bool)) private
_operatorApprovals;

    // Token name
    string private _name ;

    // Token symbol
    string private _symbol ;

    // contract owner （private or public)
    address private _owner;

    //transferOwnership は搭載しない。ハッキングによりオーナーシップが流出すると止めよう
がない。
    //デプロイした
    // onlyOwner modifier
    modifier onlyOwner() {
        require(msg.sender == _owner);
        _;
    }

    // Optional mapping for token URIs
    mapping (uint256 => string) private _tokenURIs;

    // Base URI
    string private _baseURI;

    /*
     *     bytes4(keccak256('balanceOf(address)')) == 0x70a08231
     *     bytes4(keccak256('ownerOf(uint256)')) == 0x6352211e
     *     bytes4(keccak256('approve(address,uint256)')) == 0x095ea7b3
     *     bytes4(keccak256('getApproved(uint256)')) == 0x081812fc
     *     bytes4(keccak256('setApprovalForAll(address,bool)')) == 0xa22cb465
     *     bytes4(keccak256('isApprovedForAll(address,address)')) == 0xe985e9c5
     *     bytes4(keccak256('transferFrom(address,address,uint256)')) ==
0x23b872dd
     *     bytes4(keccak256('safeTransferFrom(address,address,uint256)')) ==
0x42842e0e
     *     bytes4(keccak256('safeTransferFrom(address,address,uint256,bytes)'))
== 0xb88d4fde
     *
     *     => 0x70a08231 ^ 0x6352211e ^ 0x095ea7b3 ^ 0x081812fc ^
     *        0xa22cb465 ^ 0xe985e9c ^ 0x23b872dd ^ 0x42842e0e ^
0xb88d4fde == 0x80ac58cd
     */
    bytes4 private constant _INTERFACE_ID_ERC721 = 0x80ac58cd;

    /*
     *     bytes4(keccak256('name()')) == 0x06fdde03
     *     bytes4(keccak256('symbol()')) == 0x95d89b41
     *     bytes4(keccak256('tokenURI(uint256)')) == 0xc87b56dd
     *
     *     => 0x06fdde03 ^ 0x95d89b41 ^ 0xc87b56dd == 0x5b5e139f
     */
    bytes4 private constant _INTERFACE_ID_ERC721_METADATA = 0x5b5e139f;

    /*
     *     bytes4(keccak256('totalSupply()')) == 0x18160ddd
     *     bytes4(keccak256('tokenOfOwnerByIndex(address,uint256)')) ==
0x2f745c59
     *     bytes4(keccak256('tokenByIndex(uint256)')) == 0x4f6ccce7
     *
     *     => 0x18160ddd ^ 0x2f745c59 ^ 0x4f6ccce7 == 0x780e9d63
     */
    bytes4 private constant _INTERFACE_ID_ERC721_ENUMERABLE =
0x780e9d63;
```

```
    /**
     * @dev Initializes the contract by setting a `name` and a `symbol` to the
token collection.
     */
    constructor (string memory name, string memory symbol , string memory
baseURI ) public {

        // Token name , symbol , baseURI
        name = "0705ERC721-otpGenerator"  ;
        symbol = "0704ERC721OTPG"  ;
        baseURI = "github.com/NZRI-AZRI/";


        //Write name,symbol,owner,baseURI.
        _name = name;
        _symbol = symbol;
        _baseURI = baseURI;
        _owner = msg.sender;


        // register the supported interfaces to conform to ERC721 via ERC165
        _registerInterface(_INTERFACE_ID_ERC721);
        _registerInterface(_INTERFACE_ID_ERC721_METADATA);
        _registerInterface(_INTERFACE_ID_ERC721_ENUMERABLE);
    }

    /**
     * @dev See {IERC721-balanceOf}.
     */
    function balanceOf(address owner) public view override returns (uint256) {
        require(owner != address(0), "ERC721: balance query for the zero
address");
        return _holderTokens[owner].length();
    }

    /**
     * @dev See {IERC721-ownerOf}.
     */
    function ownerOf(uint256 tokenId) public view override returns (address) {
        return _tokenOwners.get(tokenId, "ERC721: owner query for nonexistent
token");
    }

    /**
     * @dev See {IERC721Metadata-name}.
     */
    function name() public view override returns (string memory) {
        return _name;
    }

    /**
     * @dev See {IERC721Metadata-symbol}.
     */
    function symbol() public view override returns (string memory) {
        return _symbol;
    }

    /**
     * @dev See {IERC721Metadata-tokenURI}.
     */
    function tokenURI(uint256 tokenId) public view override returns (string
memory) {
        require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent
token");

        string memory _tokenURI = _tokenURIs[tokenId];

        // If there is no base URI, return the token URI.
        if (bytes(_baseURI).length == 0) {
            return _tokenURI;
        }
        // If both are set, concatenate the baseURI and tokenURI (via
abi.encodePacked).
        if (bytes(_tokenURI).length > 0) {
            return string(abi.encodePacked(_baseURI, _tokenURI));
        }
        // If there is a baseURI but no tokenURI, concatenate the tokenID to the
baseURI.
```

```
        return string(abi.encodePacked(_baseURI, tokenId.toString()));
    }

    /**
    * @dev Returns the base URI set via {_setBaseURI}. This will be
    * automatically added as a prefix in {tokenURI} to each token's URI, or
    * to the token ID if no specific URI is set for that token ID.
    */
    function baseURI() public view returns (string memory) {
        return _baseURI;
    }

    /**
     * @dev See {IERC721Enumerable-tokenOfOwnerByIndex}.
     */
    function tokenOfOwnerByIndex(address owner, uint256 index) public view
override returns (uint256) {
        return _holderTokens[owner].at(index);
    }

    /**
     * @dev See {IERC721Enumerable-totalSupply}.
     */
    function totalSupply() public view override returns (uint256) {
        // _tokenOwners are indexed by tokenIds, so .length() returns the number
of tokenIds
        return _tokenOwners.length();
    }

    /**
     * @dev See {IERC721Enumerable-tokenByIndex}.
     */
    function tokenByIndex(uint256 index) public view override returns (uint256) {
        (uint256 tokenId, ) = _tokenOwners.at(index);
        return tokenId;
    }

    /**
     * @dev See {IERC721-approve}.
     */
    function approve(address to, uint256 tokenId) public virtual override {
        address owner = ownerOf(tokenId);
        require(to != owner, "ERC721: approval to current owner");

        require(_msgSender() == owner || isApprovedForAll(owner,
_msgSender()),
            "ERC721: approve caller is not owner nor approved for all"
        );

        _approve(to, tokenId);
    }

    /**
     * @dev See {IERC721-getApproved}.
     */
    function getApproved(uint256 tokenId) public view override returns (address)
{
        require(_exists(tokenId), "ERC721: approved query for nonexistent token");

        return _tokenApprovals[tokenId];
    }

    /**
     * @dev See {IERC721-setApprovalForAll}.
     */
    function setApprovalForAll(address operator, bool approved) public virtual
override {
        require(operator != _msgSender(), "ERC721: approve to caller");

        _operatorApprovals[_msgSender()][operator] = approved;
        emit ApprovalForAll(_msgSender(), operator, approved);
    }

    /**
     * @dev See {IERC721-isApprovedForAll}.
     */
    function isApprovedForAll(address owner, address operator) public view
override returns (bool) {
        return _operatorApprovals[owner][operator];
    }
    /**
     * @dev See {IERC721-transferFrom}.
     */
    function transferFrom(address from, address to, uint256 tokenId) public virtual
override {
        //solhint-disable-next-line max-line-length
        require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer
caller is not owner nor approved");

        _transfer(from, to, tokenId);
    }

    /**
     * @dev See {IERC721-safeTransferFrom}.
     */
    function safeTransferFrom(address from, address to, uint256 tokenId) public
virtual override {
        safeTransferFrom(from, to, tokenId, "");
    }

    /**
     * @dev See {IERC721-safeTransferFrom}.
     */
    function safeTransferFrom(address from, address to, uint256 tokenId, bytes
memory _data) public virtual override {
        require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer
caller is not owner nor approved");
        _safeTransfer(from, to, tokenId, _data);
    }

    /**
     * @dev Safely transfers `tokenId` token from `from` to `to`, checking first
that contract recipients
     * are aware of the ERC721 protocol to prevent tokens from being forever
locked.
     *
     * `_data` is additional data, it has no specified format and it is sent in call to
`to`.
```

```
     *
     * This internal function is
 equivalent to {safeTransferFrom}, and can be used to e.g.
     * implement alternative mecanisms to perform token transfer, such as
signature-based.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `tokenId` token must exist and be owned by `from`.
     * - If `to` refers to a smart contract, it must implement {IERC721Receiver-
onERC721Received}, which is called upon a safe transfer.
     *
     * Emits a {Transfer} event.
     */
    function _safeTransfer(address from, address to, uint256 tokenId, bytes
memory _data) internal virtual {
        _transfer(from, to, tokenId);
        require(_checkOnERC721Received(from, to, tokenId, _data), "ERC721:
transfer to non ERC721Receiver implementer");
    }

    /**
     * @dev Returns whether `tokenId` exists.
     *
     * Tokens can be managed by their owner or approved accounts via {approve}
or {setApprovalForAll}.
     *
     * Tokens start existing when they are minted (`_mint`),
     * and stop existing when they are burned (`_burn`).
     */
    function _exists(uint256 tokenId) internal view returns (bool) {
        return _tokenOwners.contains(tokenId);
    }

    /**
     * @dev Returns whether `spender` is allowed to manage `tokenId`.
     *
     * Requirements:
```

```
     *
     * - `tokenId` must exist.
     */
    function _isApprovedOrOwner(address spender, uint256 tokenId) internal view
returns (bool) {
        require(_exists(tokenId), "ERC721: operator query for nonexistent token");
        address owner = ownerOf(tokenId);
        return (spender == owner || getApproved(tokenId) == spender ||
isApprovedForAll(owner, spender));
    }

    /**
     * @dev Safely mints `tokenId` and transfers it to `to`.
     *
     * Requirements:
     d*
     * - `tokenId` must not exist.
     * - If `to` refers to a smart contract, it must implement {IERC721Receiver-
onERC721Received}, which is called upon a safe transfer.
     *
     * Emits a {Transfer} event.
     */
    function _safeMint(address to, uint256 tokenId) internal virtual {
        _safeMint(to, tokenId, "");
    }

    /**
     * @dev Same as {xref-ERC721-_safeMint-address-uint256-}[`_safeMint`],
with an additional `data` parameter which is
     * forwarded in {IERC721Receiver-onERC721Received} to contract recipients.
     */
    function _safeMint(address to, uint256 tokenId, bytes memory _data) internal
virtual {
        _mint(to, tokenId);
        require(_checkOnERC721Received(address(0), to, tokenId, _data),
"ERC721: transfer to non ERC721Receiver implementer");
    }

    /**
     * @dev Mints `tokenId` and transfers it to `to`.
     *
     * WARNING: Usage of this method is discouraged, use {_safeMint} whenever
possible
     *
     * Requirements:
     *
     * - `tokenId` must not exist.
     * - `to` cannot be the zero address.
     *
     * Emits a {Transfer} event.
     */
    function _mint(address to, uint256 tokenId) internal virtual {
        require(to != address(0), "ERC721: mint to the zero address");
        require(!_exists(tokenId), "ERC721: token already minted");

        _beforeTokenTransfer(address(0), to, tokenId);

        _holderTokens[to].add(tokenId);

        _tokenOwners.set(tokenId, to);

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


    /**
     * @dev Destroys `tokenId`.
     * The approval is cleared when the token is burned.
     *
     * Requirements:
     *
     * - `tokenId` must exist.
     *
     * Emits a {Transfer} event.
     */
     //intenal関数であってコントラクト外部のユーザーやオーナーは呼び出せない。発動しない関
数。
    function _burn(uint256 tokenId) internal virtual {
        address owner = ownerOf(tokenId);
        _beforeTokenTransfer(owner, address(0), tokenId);

        // Clear approvals
        _approve(address(0), tokenId);

        // Clear metadata (if any)
        if (bytes(_tokenURIs[tokenId]).length != 0) {
            delete _tokenURIs[tokenId];
        }

        _holderTokens[owner].remove(tokenId);

        _tokenOwners.remove(tokenId);

        emit Transfer(owner, address(0), tokenId);
    }


    /**
     * @dev Transfers `tokenId` from `from` to `to`.
     *  As opposed to {transferFrom}, this imposes no restrictions on msg.sender.
     *
     * Requirements:
     *
     * - `to` cannot be the zero address.
     * - `tokenId` token must be owned by `from`.
     *
     * Emits a {Transfer} event.
     */
    function _transfer(address from, address to, uint256 tokenId) internal virtual {
        require(ownerOf(tokenId) == from, "ERC721: transfer of token that is not
own");
        require(to != address(0), "ERC721: transfer to the zero address");

        _beforeTokenTransfer(from, to, tokenId);

        // Clear approvals from the previous owner
        _approve(address(0), tokenId);

        _holderTokens[from].remove(tokenId);
        _holderTokens[to].add(tokenId);

        _tokenOwners.set(tokenId, to);

        emit Transfer(from, to, tokenId);
    }

    /**
     * @dev Sets `_tokenURI` as the tokenURI of `tokenId`.
     *
     * Requirements:
     *
     * - `tokenId` must exist.
     */
    function _setTokenURI(uint256 tokenId, string memory _tokenURI) internal
virtual {
        require(_exists(tokenId), "ERC721Metadata: URI set of nonexistent token");
        _tokenURIs[tokenId] = _tokenURI;
    }

    /**
     * @dev Internal function to set the base URI for all token IDs. It is
     * automatically added as a prefix to the value returned in {tokenURI},
     * or to the token ID if {tokenURI} is empty.
     */
    function _setBaseURI(string memory baseURI_) internal virtual {
        _baseURI = baseURI_;
    }

    /**
     * @dev Internal function to invoke {IERC721Receiver-onERC721Received} on
a target address.
     * The call is not executed if the target address is not a contract.
     *
     * @param from address representing the previous owner of the given token
ID
     * @param to target address that will receive the tokens
     * @param tokenId uint256 ID of the token to be transferred
     * @param _data bytes optional data to send along with the call
     * @return bool whether the call correctly returned the expected magic value
     */
    function _checkOnERC721Received(address from, address to, uint256 tokenId,
bytes memory _data)
        private returns (bool)
    {
        if (!to.isContract()) {
            return true;
        }
        bytes memory returndata = to.functionCall(abi.encodeWithSelector(
            IERC721Receiver(to).onERC721Received.selector,
            _msgSender(),
            from,
            tokenId,
            _data
        ), "ERC721: transfer to non ERC721Receiver implementer");
        bytes4 retval = abi.decode(returndata, (bytes4));
        return (retval == _ERC721_RECEIVED);
    }

    function _approve(address to, uint256 tokenId) private {
        _tokenApprovals[tokenId] = to;
        emit Approval(ownerOf(tokenId), to, tokenId);
    }

    /**
     * @dev Hook that is called before any token transfer. This includes minting
     * and burning.
     *
     * Calling conditions:
     *
     * - When `from` and `to` are both non-zero, ``from``'s `tokenId` will be
     * transferred to `to`.
     * - When `from` is zero, `tokenId` will be minted for `to`.
     * - When `to` is zero, ``from``'s `tokenId` will be burned.
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     *
     * To learn more about hooks, head to xref:ROOT:extending-
contracts.adoc#using-hooks[Using Hooks].
     */
    function _beforeTokenTransfer(address from, address to, uint256 tokenId)
internal virtual { }

//ERC721OTP==================================================
==============

///////////////////////////NFT の所持者確認部分////////////////////////////////////

    //この関数が true とみなすトークン持ち主の処理を続行
    function requireNftOwner(uint256 _tokenId) public view returns(bool){
        require(msg.sender != address(0), "ERC721: balance query for the zero
address");
        require(ownerOf(_tokenId) == msg.sender  );//tokenId のオーナーが関数実行者
か確認

        return true ;
    }
//////////////////////////////////////////////////////////////




    //NFT CARD DATA

    struct PlayingCard
    {
        /*ガスコスト削減のためストレージ 1 スロット＝bytes32 に収めること。*/

          //トランプカード情報 2bytes
          bytes1 cardSuits;
          bytes1 cardNumber;

          //トレカ的情報 全て bytes1、uint8 の 255 までの値。5bytes
          bytes1 cardType;//カードタイプ。例えばじゃんけんゲーム用の値
          bytes1 cardA;//攻撃力など
          bytes1 cardB;//防御力など
          bytes1 cardC;//特殊攻撃、賢さ、魔術攻撃力など
          bytes1 cardD;//特殊防御、感性、魔術防御力など

          //追記項目、あるいはワンタイムパスワード失効の有無など。今回は何も書かない。運営か
ら切符を切れる関数を設定する場合この変数にアクセス
```

```
        bytes4 cardEx;//他の項目など

        //残バイト部分。創作者 ID もしくは文字。4byte の創作者ティッカーシンボル
        bytes4 cardCreator;//製造者番号　例 utf-8; " NZRI" = 0x4E5A5249(Hex) =
1314542153(uint)
        bytes16 cardData;//シリアル ID でもあるランダムデータ。場合によってはゲームで使
う変数。
    }



    //mapping struct PlayingCard
    mapping (uint256 => PlayingCard ) public idToPlayingCard;
    //example code is ; // idToPlayingCard[_tokenId] = playingCard (_suit ,
_number  , _type, _a1 , _b2 , _c3 ,_d4 , _ex , _creator,_data);




    /**
    * Creator Data Section====================
    */
    //Creator data　作者の名前とウェブサイトやメールアドレス、ＳＮＳアカウント、電話番号な
どを記載する。
    //(有事の際はこのサイトでコントラクトの運用に関してアナウンスする)

    //Site　招待するウェブサイトアドレス。この部分は運営であるこちらで入力できるようにさせて
いただきたい。
    //address setter getter
    function setSiteBaseURI(string memory _newAddress) public onlyOwner {
        _setBaseURI(_newAddress);//erc721 搭載の機能にも保存
    }

//==================================================
=====================
   /**
    * oneTimePassCode Secret Section====================
    * gene と auth で統一すること。一致していないと動作できない。共通秘密シード
    */
    //secret private
    string private _secret = "NZRI ワンタイムパスワードトークンクラウドファンディング第一
弾テスト 3";//examle.
```

```
    //contract-address-secret
    address private _secAdr = 0x0f398803BE4319B98F164cae47589797aC5cF906
;//authではgeneのコントラクトアドレス使う
    //この_secAdrで他のコントラクトとこのコントラクトを区別する。
    function setThisAdr(address _newAdr) public onlyOwner {
        _secAdr  = _newAdr ;
    }

    /*このほかにいくつかbytes32やbool等のシークレットを追加してワンタイムパスワードシー
ドに追加してもよい */
    bytes32 private byt1 =
0x51f3d69596f11b3deba29c805ee8094f615ff8ee0d78779ca3944661d9385c92;


    //CreatorOneTimeNum--->controlOTnum--->ctrOtNum
    uint256 private _ctrOtNum = 202107230808;//geneとauthで統一すること
    //クリエイター・オーナーが任意の時に任意の番号をランダムオラクル供給するための関数。ラン
ダムはJavascriptの関数などで生成する。
    function setOtpNum(uint256 _tokenId , uint256 _newNum) public onlyOwner
{
        require( requireNftOwner(_tokenId) == true );
        _ctrOtNum  = _newNum ;
    }

    //固定TP 持つ人のアドレスとトークンIDによって変わる。以下のコンスタント番号をつかう。
（オリンピックの開催予定日）
    uint256 private _constantNum = 2021072308082020;

//トークン券面ID番号のみを使う、固定パスワード
TP=============================================
==

    //番号が固定の為変わらないパスワード。トークン固有パスワードTP
    //あるIDのトークンを持つ人は誰でも見れる。トークンが流通するときこのパスワードを知って
いる人は複数現れる。
    function getTkpw(uint256 _tokenId) public view returns (bytes32) {
        require( requireNftOwner(_tokenId) == true );
        //条件通りならばクリエイターのセットしたパスワードをリターンする
        return sha256(abi.encodePacked( _constantNum , _secret, _tokenId,
_secAdr));
    }
    //7 桁 TP
    function getTkpw7Num(uint256 _tokenId) public view returns (uint256) {
        bytes32 otpByte = getTkpw( _tokenId) ;//msgsender が bytes32 型 TOTP を呼
べるか調べる。呼び出し元 NG ならば require 通れない。
        uint256 otpUint = uint256(otpByte);//bytes32 を uint 整数に変換。その整数の 7
桁を使用する。
        return otpUint % 10**7; //10000000 の剰余を返す。7 桁表示の為。
    }

//================================疑似ランダムシード token totp
    function getTkTotpRn(uint256 _tokenId) public view returns (bytes32) {
        require( requireNftOwner(_tokenId) == true );
        require( otpMp > 0 );//otpMp>0 を要求

        uint256 bnVar = block.number ;    //現在 blocknumber 取得
        uint256 bnModMp = block.number % otpMp ;    //blocknumber の剰余を出す。
        while(bnModMp > 0) { //条件式が満たされている場合、繰り返し処理が続く。
            bnModMp = bnModMp - 1;
            bnVar = bnVar - 1 ;
        }
        uint256 imRandom= uint(  sha256(abi.encodePacked(_secAdr,
blockhash(bnVar) )) );
        return sha256(abi.encodePacked( bnVar, _constantNum , _secret, _tokenId,
_secAdr, imRandom ));
    }
    //7 桁 TOTP
    function getTkTotpRn7Num(uint256 _tokenId) public view returns (uint256) {
        bytes32 otpByte = getTkTotpRn(_tokenId) ;//msgsender が bytes32 型 TOTP
を呼べるか調べる。呼び出し元 NG ならば require 通れない。
        uint256 otpUint = uint256(otpByte);//bytes32 を uint 整数に変換。その整数の 7
桁を使用する。
        return otpUint % 10**7; //10000000 の剰余を返す。7 桁表示の為。
    }
//================================


//顧客アドレス依存型の定数変更型
```

```
OTP=====================================================
===
    //getYourConstOTP() それぞれの人が持つパスワードを表示させる。任意の時間に数字が変
わって OTP 変わることがあるやつ。
    //固定の為、紙印刷チケット用のパスワードに使う。
    function getConstOtp(uint256 _tokenId) public view returns (bytes32) {
        require( requireNftOwner(_tokenId) == true );
        //条件通りならばクリエイターのセットしたパスワードをリターンする
        return sha256(abi.encodePacked(byt1, _constantNum, _secret, msg.sender,
_tokenId, _secAdr));
    }
    //7 桁 OTP
    function getConstOtp7Num(uint256 _tokenId) public view returns (uint256) {
        bytes32 otpByte = getConstOtp( _tokenId) ;//msgsender が bytes32 型 TOTP
を呼べるか調べる。呼び出し元 NG ならば require 通れない。
        uint256 otpUint = uint256(otpByte);//bytes32 を uint 整数に変換。その整数の 7
桁を使用する。
        return otpUint % 10**7; //10000000 の剰余を返す。7 桁表示の為。
    }

//オーナー変更型
OTP=====================================================
===
    //getYourOTP() それぞれの人が持つパスワードを表示させる。任意の時間に数字が変わって
OTP 変わることがあるやつ。
    //ブロックチェーン外のトークン保有者が好き放題にパスワードを変えられるモード
    function getOtp(uint256 _tokenId) public view returns (bytes32) {
        require( requireNftOwner(_tokenId) == true );
        //条件通りならばクリエイターのセットしたパスワードをリターンする
        return sha256(abi.encodePacked(byt1, _ctrOtNum, _secret, msg.sender,
_tokenId, _secAdr));
    }

    //7 桁 OTP
    function getOtp7Num(uint256 _tokenId) public view returns (uint256) {
        bytes32 otpByte = getOtp( _tokenId) ;//msgsender が bytes32 型 TOTP を呼べ
るか調べる。呼び出し元 NG ならば require 通れない。
        uint256 otpUint = uint256(otpByte);//bytes32 を uint 整数に変換。その整数の 7
桁を使用する。
        return otpUint % 10**7; //10000000 の剰余を返す。7 桁表示の為。
    }

//時刻同期型
OTP、TOTP=================================================
=======
    //oneTime-multiple 初期値 3、【重要】; auth と同期させること。45 秒で変わる OTP.

    //具体的には現在のブロック番号が 1 で割れれば全て。2 で割れれば偶数の時、n で割って剰余が
いくつかで対応を決める。
    //multiple この変数を変えて 15 の n 倍の秒数を設定する。例としてこれが 2 の時 2*15sec =
30sec にする。
    uint8 public otpMp = 3 ;//!初期値をゼロにしない。
    //setter
    function setOtpMp(uint8 _newMp) public onlyOwner {
        require( otpMp > 0 );//ゼロでないか確認。ゼロ以上か確認。
        otpMp  = _newMp ;
    }
//==============================================
    function getTotp(uint256 _tokenId) public view returns (bytes32) {
        require( requireNftOwner(_tokenId) == true );
        require( otpMp > 0 );//otpMp>0 を要求
        //変数定義
        uint256 bnVar = block.number ;    //現在 blocknumber 取得
        uint256 bnModMp = block.number % otpMp ;    //blocknumber の剰余を出す。
        //while 文で計算する
        //bn256Mp がゼロになり mod ゼロになるブロックナンバーまで処理を続ける。ゼロになる
まで bnVar を戻らせる
        while(bnModMp > 0) { //条件式が満たされている場合、繰り返し処理が続く。
            bnModMp = bnModMp - 1;
            bnVar = bnVar - 1 ;
        }
        //bnModMp がゼロになるまで bnVar 引き算してワンタイムパスワードに利用する
        return sha256(abi.encodePacked(byt1, _ctrOtNum, bnVar, _secret,
msg.sender, _tokenId, _secAdr));
    }
    //7 桁 TOTP
    function getTotp7Num(uint256 _tokenId) public view returns (uint256) {
        bytes32 otpByte = getTotp(_tokenId) ;//msgsender が bytes32 型 TOTP を呼べ
るか調べる。呼び出し元 NG ならば require 通れない。
        uint256 otpUint = uint256(otpByte);//bytes32 を uint 整数に変換。その整数の 7
```

```
桁を使用する。
        return otpUint % 10**7; //10000000 の剰余を返す。7 桁表示の為。
    }
//================================疑似ランダムシード totp
    function getTotpRn(uint256 _tokenId) public view returns (bytes32) {
        require( requireNftOwner(_tokenId) == true );
        require( otpMp > 0 );//otpMp>0 を要求。そして 256 以内であること。
( otpMp<256 ) uint8 なのでこの条件は満たしているはず。

        //変数定義
        uint256 bnVar = block.number ;    //現在 blocknumber 取得
        uint256 bnModMp = block.number % otpMp ;    //blocknumber の剰余を出す。
        //while 文で計算する
        //bnModMp がゼロになり mod ゼロになるブロックナンバーまで bnVar のデクリメント処
理を続ける。ゼロになるまで bnVar を戻らせる
        while(bnModMp > 0) { //条件式が満たされている場合、繰り返し処理が続く。
            bnModMp = bnModMp - 1;
            bnVar = bnVar - 1 ;
        }

        //bnVar と bnVar のブロック番号の時のハッシュを疑似 randam とする。
        //bnVar の blockNumber に対応したブロックハッシュを呼ぶ。これは 256 ブロック以内
を参照。
        uint256 imRandom= uint(  sha256(abi.encodePacked(_secAdr,
blockhash(bnVar) )) );


        //ブロックハッシュ要素を引数に含んだワンタイムパスワードランダムやクリエイター番号を
追加した場合。（フルコース OTP）
        return sha256(abi.encodePacked(byt1, _ctrOtNum, bnVar, _secret,
msg.sender, _tokenId, _secAdr, imRandom ));
    }
    //7 桁 TOTP
    function getTotpRn7Num(uint256 _tokenId) public view returns (uint256) {
        bytes32 otpByte = getTotpRn(_tokenId) ;//msgsender が bytes32 型 TOTP を
呼べるか調べる。呼び出し元 NG ならば require 通れない。
        uint256 otpUint = uint256(otpByte);//bytes32 を uint 整数に変換。その整数の 7
桁を使用する。
        return otpUint % 10**7; //10000000 の剰余を返す。7 桁表示の為。
    }
```

```
//==================================

//burn() is removed.
//only mint().

  /**
   * @dev Mints a new NFT.
   * @param _to The address that will own the minted NFT.
   * @param _tokenId of the NFT to be minted by the msg.sender.
   *
   */
  function mint(
    address _to,
    uint256 _tokenId,
    //トランプカードのスート、カードの番号 bytes1=0x00-0xff
    bytes1 _suit,
    bytes1 _number,
    //三すくみ型じゃんけんゲーム用タイプ数字
    //なお単純にじゃんけんするとき、_type を筆頭に_a1-_d4 までの値を 3 で割った剰余値から
じゃんけんの手を決めてもよい。
    //a1 から d4 までの値から 4 つのじゃんけんの所有を表せる。
    bytes1 _type,
    bytes1 _a1,
    bytes1 _b2,
    bytes1 _c3,
    bytes1 _d4,
    bytes4 _ex,
    //トークンシリアル情報、創作者情報
    bytes4 _creator,
    bytes16 _data
  )
    external
    onlyOwner
  {
    //mint 実行 ERC721 規格部分。URI は設定しない。URI は setSite 関数とトークン ID から作
れるので無くす。
    //tokenid は 0 から 4 番までをテスト用として予約する。
    //ユーザーへ 5 番から発行する。4 が忌数字なので回避し、いいご縁の為 5 から始めるものとする。
    _mint(_to, _tokenId);
    //mint実行 OTPカードとしての部分
    //CF特典としていわゆるガチャ時に最低値と最低の値の閾値を検討し、最低値を200に引き上げる。
    //255～200までをランダム選択して発行。CF以外で売るときは255-10までにする。

    idToPlayingCard[_tokenId] = PlayingCard (_suit , _number  , _type, _a1 , _b2 , _c3 ,_d4 , _ex , _creator,_data);

  }

}


/*
    function setCardEx(uint256 _tokenId , bytes4 _exData) external onlyOwner {
        //いわゆる切符を切る処理。ワンタイムパスワードを使うサービスをユーザーが使い、あるいは期限が切れ終了した時に書き換える部分
        //SUICA的な利用方法ではチャージ残高として使えるかも。bytes4=uint8*4=uint32
        //今回はデフォルト値を0x000007d0=2000にする
        idToPlayingCard[_tokenId].cardEx = _exData ;
    }
*/

/*
  bytes1 0xfa
  bytes4 0x4E5A5249  --->>"nzri"
  bytes4 "nzri"--> uint ---->1314542153
  bytes4 0x000005dc  --->1500
  bytes8 0x4E5A52494E5A5249
  bytes16 0x4E5A52494E5A52494E5A52494E5A5249
*/
```

ABI

```
[
        {
                "inputs": [
                        {
                                "internalType": "string",
				"name": "name",
				"type": "string"
			},
			{
				"internalType": "string",
				"name": "symbol",
				"type": "string"
			},
			{
				"internalType": "string",
				"name": "baseURI",
				"type": "string"
			}
		],
		"stateMutability": "nonpayable",
		"type": "constructor"
	},
	{
		"anonymous": false,
		"inputs": [
			{
				"indexed": true,
				"internalType": "address",
				"name": "owner",
				"type": "address"
			},
			{
				"indexed": true,
				"internalType": "address",
				"name": "approved",
				"type": "address"
			},
			{
				"indexed": true,
				"internalType": "uint256",
				"name": "tokenId",
				"type": "uint256"
			}
		],
		"name": "Approval",
		"type": "event"
	},
	{
		"anonymous": false,
		"inputs": [
			{
				"indexed": true,
				"internalType": "address",
				"name": "owner",
				"type": "address"
			},
			{
				"indexed": true,
				"internalType": "address",
				"name": "operator",
				"type": "address"
			},
			{
				"indexed": false,
				"internalType": "bool",
				"name": "approved",
				"type": "bool"
			}
		],
		"name": "ApprovalForAll",
		"type": "event"
	},
	{
		"anonymous": false,
		"inputs": [
			{
				"indexed": true,
				"internalType": "address",
				"name": "from",
				"type": "address"
			},
			{
				"indexed": true,
				"internalType": "address",
				"name": "to",
            "type": "address"
        },
        {
            "indexed": true,
            "internalType": "uint256",
            "name": "tokenId",
            "type": "uint256"
        }
    ],
    "name": "Transfer",
    "type": "event"
},
{
    "inputs": [
        {
            "internalType": "address",
            "name": "to",
            "type": "address"
        },
        {
            "internalType": "uint256",
            "name": "tokenId",
            "type": "uint256"
        }
    ],
    "name": "approve",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "address",
            "name": "owner",
            "type": "address"
        }
    ],
    "name": "balanceOf",
    "outputs": [
			{
				"internalType": "uint256",
				"name": "",
				"type": "uint256"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [],
		"name": "baseURI",
		"outputs": [
			{
				"internalType": "string",
				"name": "",
				"type": "string"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "uint256",
				"name": "tokenId",
				"type": "uint256"
			}
		],
		"name": "getApproved",
		"outputs": [
			{
				"internalType": "address",
				"name": "",
				"type": "address"
			}
		],
		"stateMutability": "view",
		"type": "function"
```

```
    },
    {
        "inputs": [
            {
                "internalType": "uint256",
                "name": "_tokenId",
                "type": "uint256"
            }
        ],
        "name": "getConstOtp",
        "outputs": [
            {
                "internalType": "bytes32",
                "name": "",
                "type": "bytes32"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "uint256",
                "name": "_tokenId",
                "type": "uint256"
            }
        ],
        "name": "getConstOtp7Num",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
```

```
		"inputs": [
			{
				"internalType": "uint256",
				"name": "_tokenId",
				"type": "uint256"
			}
		],
		"name": "getOtp",
		"outputs": [
			{
				"internalType": "bytes32",
				"name": "",
				"type": "bytes32"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "uint256",
				"name": "_tokenId",
				"type": "uint256"
			}
		],
		"name": "getOtp7Num",
		"outputs": [
			{
				"internalType": "uint256",
				"name": "",
				"type": "uint256"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [
			{
```

```
                "internalType": "uint256",
                "name": "_tokenId",
                "type": "uint256"
            }
        ],
        "name": "getTkTotpRn",
        "outputs": [
            {
                "internalType": "bytes32",
                "name": "",
                "type": "bytes32"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "uint256",
                "name": "_tokenId",
                "type": "uint256"
            }
        ],
        "name": "getTkTotpRn7Num",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "uint256",
                "name": "_tokenId",
                "type": "uint256"
            }
        ],
        "name": "getTkpw",
        "outputs": [
            {
                "internalType": "bytes32",
                "name": "",
                "type": "bytes32"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "uint256",
                "name": "_tokenId",
                "type": "uint256"
            }
        ],
        "name": "getTkpw7Num",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "uint256",
                "name": "_tokenId",
                "type": "uint256"
            }
        ],
        "name": "getTotp",
        "outputs": [
            {
                "internalType": "bytes32",
                "name": "",
                "type": "bytes32"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "uint256",
                "name": "_tokenId",
                "type": "uint256"
            }
        ],
        "name": "getTotp7Num",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "uint256",
                "name": "_tokenId",
                "type": "uint256"
            }
        ],
        "name": "getTotpRn",
		"outputs": [
			{
				"internalType": "bytes32",
				"name": "",
				"type": "bytes32"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "uint256",
				"name": "_tokenId",
				"type": "uint256"
			}
		],
		"name": "getTotpRn7Num",
		"outputs": [
			{
				"internalType": "uint256",
				"name": "",
				"type": "uint256"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "uint256",
				"name": "",
				"type": "uint256"
			}
		],
		"name": "idToPlayingCard",
		"outputs": [
			{
```

```
		"internalType": "bytes1",
		"name": "cardSuits",
		"type": "bytes1"
	},
	{
		"internalType": "bytes1",
		"name": "cardNumber",
		"type": "bytes1"
	},
	{
		"internalType": "bytes1",
		"name": "cardType",
		"type": "bytes1"
	},
	{
		"internalType": "bytes1",
		"name": "cardA",
		"type": "bytes1"
	},
	{
		"internalType": "bytes1",
		"name": "cardB",
		"type": "bytes1"
	},
	{
		"internalType": "bytes1",
		"name": "cardC",
		"type": "bytes1"
	},
	{
		"internalType": "bytes1",
		"name": "cardD",
		"type": "bytes1"
	},
	{
		"internalType": "bytes4",
		"name": "cardEx",
		"type": "bytes4"
	},
	{
				"internalType": "bytes4",
				"name": "cardCreator",
				"type": "bytes4"
			},
			{
				"internalType": "bytes16",
				"name": "cardData",
				"type": "bytes16"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "address",
				"name": "owner",
				"type": "address"
			},
			{
				"internalType": "address",
				"name": "operator",
				"type": "address"
			}
		],
		"name": "isApprovedForAll",
		"outputs": [
			{
				"internalType": "bool",
				"name": "",
				"type": "bool"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "address",
				"name": "_to",
				"type": "address"
			},
			{
				"internalType": "uint256",
				"name": "_tokenId",
				"type": "uint256"
			},
			{
				"internalType": "bytes1",
				"name": "_suit",
				"type": "bytes1"
			},
			{
				"internalType": "bytes1",
				"name": "_number",
				"type": "bytes1"
			},
			{
				"internalType": "bytes1",
				"name": "_type",
				"type": "bytes1"
			},
			{
				"internalType": "bytes1",
				"name": "_a1",
				"type": "bytes1"
			},
			{
				"internalType": "bytes1",
				"name": "_b2",
				"type": "bytes1"
			},
			{
				"internalType": "bytes1",
				"name": "_c3",
				"type": "bytes1"
			},
			{
				"internalType": "bytes1",
				"name": "_d4",
				"type": "bytes1"
			},
			{
				"internalType": "bytes4",
				"name": "_ex",
				"type": "bytes4"
			},
			{
				"internalType": "bytes4",
				"name": "_creator",
				"type": "bytes4"
			},
			{
				"internalType": "bytes16",
				"name": "_data",
				"type": "bytes16"
			}
		],
		"name": "mint",
		"outputs": [],
		"stateMutability": "nonpayable",
		"type": "function"
	},
	{
		"inputs": [],
		"name": "name",
		"outputs": [
			{
				"internalType": "string",
				"name": "",
				"type": "string"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [],
		"name": "otpMp",
		"outputs": [
			{
				"internalType": "uint8",
				"name": "",
				"type": "uint8"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "uint256",
				"name": "tokenId",
				"type": "uint256"
			}
		],
		"name": "ownerOf",
		"outputs": [
			{
				"internalType": "address",
				"name": "",
				"type": "address"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "uint256",
				"name": "_tokenId",
				"type": "uint256"
			}
		],
		"name": "requireNftOwner",
		"outputs": [
			{
				"internalType": "bool",
				"name": "",
				"type": "bool"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "address",
				"name": "from",
				"type": "address"
			},
			{
				"internalType": "address",
				"name": "to",
				"type": "address"
			},
			{
				"internalType": "uint256",
				"name": "tokenId",
				"type": "uint256"
			}
		],
		"name": "safeTransferFrom",
		"outputs": [],
		"stateMutability": "nonpayable",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "address",
				"name": "from",
				"type": "address"
			},
			{
```

```
                "internalType": "address",
                "name": "to",
                "type": "address"
            },
            {
                "internalType": "uint256",
                "name": "tokenId",
                "type": "uint256"
            },
            {
                "internalType": "bytes",
                "name": "_data",
                "type": "bytes"
            }
        ],
        "name": "safeTransferFrom",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "operator",
                "type": "address"
            },
            {
                "internalType": "bool",
                "name": "approved",
                "type": "bool"
            }
        ],
        "name": "setApprovalForAll",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [
		{
			"internalType": "uint8",
			"name": "_newMp",
			"type": "uint8"
		}
	],
	"name": "setOtpMp",
	"outputs": [],
	"stateMutability": "nonpayable",
	"type": "function"
},
{
	"inputs": [
		{
			"internalType": "uint256",
			"name": "_tokenId",
			"type": "uint256"
		},
		{
			"internalType": "uint256",
			"name": "_newNum",
			"type": "uint256"
		}
	],
	"name": "setOtpNum",
	"outputs": [],
	"stateMutability": "nonpayable",
	"type": "function"
},
{
	"inputs": [
		{
			"internalType": "string",
			"name": "_newAddress",
			"type": "string"
		}
	],
	"name": "setSiteBaseURI",
	"outputs": [],
	"stateMutability": "nonpayable",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "address",
				"name": "_newAdr",
				"type": "address"
			}
		],
		"name": "setThisAdr",
		"outputs": [],
		"stateMutability": "nonpayable",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "bytes4",
				"name": "interfaceId",
				"type": "bytes4"
			}
		],
		"name": "supportsInterface",
		"outputs": [
			{
				"internalType": "bool",
				"name": "",
				"type": "bool"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [],
		"name": "symbol",
		"outputs": [
			{
				"internalType": "string",
				"name": "",
				"type": "string"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "uint256",
				"name": "index",
				"type": "uint256"
			}
		],
		"name": "tokenByIndex",
		"outputs": [
			{
				"internalType": "uint256",
				"name": "",
				"type": "uint256"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "address",
				"name": "owner",
				"type": "address"
			},
			{
				"internalType": "uint256",
				"name": "index",
				"type": "uint256"
			}
		],
		"name": "tokenOfOwnerByIndex",
		"outputs": [
			{
				"internalType": "uint256",
				"name": "",
				"type": "uint256"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "uint256",
				"name": "tokenId",
				"type": "uint256"
			}
		],
		"name": "tokenURI",
		"outputs": [
			{
				"internalType": "string",
				"name": "",
				"type": "string"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [],
		"name": "totalSupply",
		"outputs": [
			{
				"internalType": "uint256",
				"name": "",
				"type": "uint256"
			}
		],
		"stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "address",
                "name": "from",
                "type": "address"
            },
            {
                "internalType": "address",
                "name": "to",
                "type": "address"
            },
            {
                "internalType": "uint256",
                "name": "tokenId",
                "type": "uint256"
            }
        ],
        "name": "transferFrom",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
    }
]
```