```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

import "@openzeppelin/contracts@4.7.3/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts@4.7.3/token/ERC20/extensions/ERC20Burnable.sol";
import "@openzeppelin/contracts@4.7.3/token/ERC20/extensions/ERC20Snapshot.sol";
import "@openzeppelin/contracts@4.7.3/access/Ownable.sol";
import "@openzeppelin/contracts@4.7.3/token/ERC20/extensions/draft-ERC20Permit.sol";
import "@openzeppelin/contracts@4.7.3/token/ERC20/extensions/ERC20Votes.sol";

contract XLYNX is ERC20, ERC20Burnable, ERC20Snapshot, Ownable, ERC20Permit, ERC20Votes {
 bool private isBurningEnabled = true;

 constructor() ERC20("X-LYNX", "LYNS") ERC20Permit("X-LYNX") {
 _mint(msg.sender, 2100000000 * 10 ** decimals());
 }

function snapshot() public onlyOwner {
 _snapshot();
 }

function toggleBurn(bool enable) public onlyOwner {
 isBurningEnabled = enable;
 }

function _transfer(address sender, address recipient, uint256 amount) internal override {
 uint256 burnAmount = 0;
 if (isBurningEnabled) {
 burnAmount = (amount * 2) / 100; // 2% of the transaction amount
 _burn(sender, burnAmount);
 }
 uint256 transferAmount = amount - burnAmount;


 require(amount == transferAmount + burnAmount, "Burn value invalid");


 super._transfer(sender, recipient, transferAmount);
 }


 // The following functions are overrides required by Solidity.


 function _beforeTokenTransfer(address from, address to, uint256 amount) internal override(ERC20, ERC20Snapshot) {
 super._beforeTokenTransfer(from, to, amount);
 }


 function _afterTokenTransfer(address from, address to, uint256 amount) internal override(ERC20, ERC20Votes) {
 super._afterTokenTransfer(from, to, amount);
 }


function _mint(address to, uint256 amount) internal override(ERC20, ERC20Votes) {
 super._mint(to, amount);
}


function _burn(address account, uint256 amount) internal override(ERC20, ERC20Votes) {
 super._burn(account, amount);
}
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.5.0) (token/ERC20/extensions/ERC20Votes.sol)


pragma solidity ^0.8.0;
```

```
import "./draft-ERC20Permit.sol";
import "../../../utils/math/Math.sol";
import "../../../governance/utils/IVotes.sol";
import "../../../utils/math/SafeCast.sol";
import "../../../utils/cryptography/ECDSA.sol";

/**
 * @dev Extension of ERC20 to support Compound-like voting and delegation. This version is more generic than Compound's,
 * and supports token supply up to 2^224^ - 1, while COMP is limited to 2^96^ - 1.
 *
 * NOTE: If exact COMP compatibility is required, use the {ERC20VotesComp} variant of this module.
 *
 * This extension keeps a history (checkpoints) of each account's vote power. Vote power can be delegated either
 * by calling the {delegate} function directly, or by providing a signature to be used with {delegateBySig}. Voting
 * power can be queried through the public accessors {getVotes} and {getPastVotes}.
 *
 * By default, token balance does not account for voting power. This makes transfers cheaper. The downside is that it
 * requires users to delegate to themselves in order to activate checkpoints and have their voting power tracked.
 *
 * _Available since v4.2._
 */
abstract contract ERC20Votes is IVotes, ERC20Permit {
    struct Checkpoint {
        uint32 fromBlock;
        uint224 votes;
    }

    bytes32 private constant _DELEGATION_TYPEHASH =
        keccak256("Delegation(address delegatee,uint256 nonce,uint256 expiry)");

    mapping(address => address) private _delegates;
    mapping(address => Checkpoint[]) private _checkpoints;
    Checkpoint[] private _totalSupplyCheckpoints;

    /**
     * @dev Get the `pos`-th checkpoint for `account`.
     */
    function checkpoints(address account, uint32 pos) public view virtual returns (Checkpoint memory) {
        return _checkpoints[account][pos];
    }

    /**
     * @dev Get number of checkpoints for `account`.
     */
    function numCheckpoints(address account) public view virtual returns (uint32) {
        return SafeCast.toUint32(_checkpoints[account].length);
    }

    /**
     * @dev Get the address `account` is currently delegating to.
     */
    function delegates(address account) public view virtual override returns (address) {
        return _delegates[account];
    }

    /**
     * @dev Gets the current votes balance for `account`
     */
    function getVotes(address account) public view virtual override returns (uint256) {
        uint256 pos = _checkpoints[account].length;
        return pos == 0 ? 0 : _checkpoints[account][pos - 1].votes;
    }

    /**
     * @dev Retrieve the number of votes for `account` at the end of `blockNumber`.
     *
     * Requirements:
     *
     * - `blockNumber` must have been already mined
     */
    function getPastVotes(address account, uint256 blockNumber) public view virtual override returns
(uint256) {
        require(blockNumber < block.number, "ERC20Votes: block not yet mined");
        return _checkpointsLookup(_checkpoints[account], blockNumber);
    }

    /**
     * @dev Retrieve the `totalSupply` at the end of `blockNumber`. Note, this value is the sum of all
balances.
     * It is but NOT the sum of all the delegated votes!
     *
     * Requirements:
     *
     * - `blockNumber` must have been already mined
     */
    function getPastTotalSupply(uint256 blockNumber) public view virtual override returns (uint256) {
        require(blockNumber < block.number, "ERC20Votes: block not yet mined");
        return _checkpointsLookup(_totalSupplyCheckpoints, blockNumber);
    }
    /**
     * @dev Lookup a value in a list of (sorted) checkpoints.
     */
    function _checkpointsLookup(Checkpoint[] storage ckpts, uint256 blockNumber) private view
returns (uint256) {
        // We run a binary search to look for the earliest checkpoint taken after `blockNumber`.
        //
        // During the loop, the index of the wanted checkpoint remains in the range [low-1, high).
        // With each iteration, either `low` or `high` is moved towards the middle of the range to
maintain the invariant.
        // - If the middle checkpoint is after `blockNumber`, we look in [low, mid)
        // - If the middle checkpoint is before or equal to `blockNumber`, we look in [mid+1, high)
        // Once we reach a single value (when low == high), we've found the right checkpoint at the
index high-1, if not
        // out of bounds (in which case we're looking too far in the past and the result is 0).
        // Note that if the latest checkpoint available is exactly for `blockNumber`, we end up with an
index that is
        // past the end of the array, so we technically don't find a checkpoint after `blockNumber`, but it
works out
        // the same.
        uint256 high = ckpts.length;
        uint256 low = 0;
        while (low < high) {
            uint256 mid = Math.average(low, high);
            if (ckpts[mid].fromBlock > blockNumber) {
                high = mid;
            } else {
                low = mid + 1;
            }
        }

        return high == 0 ? 0 : ckpts[high - 1].votes;
    }
    /**
     * @dev Delegate votes from the sender to `delegatee`.
     */
    function delegate(address delegatee) public virtual override {
        _delegate(_msgSender(), delegatee);
    }

    /**
     * @dev Delegates votes from signer to `delegatee`
     */
    function delegateBySig(
        address delegatee,
        uint256 nonce,
        uint256 expiry,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) public virtual override {
        require(block.timestamp <= expiry, "ERC20Votes: signature expired");
        address signer = ECDSA.recover(
            _hashTypedDataV4(keccak256(abi.encode(_DELEGATION_TYPEHASH, delegatee, nonce, expiry))),
            v,
            r,
            s
        );
        require(nonce == _useNonce(signer), "ERC20Votes: invalid nonce");
        _delegate(signer, delegatee);
    }
```

```
/**
 * @dev Maximum token supply. Defaults to `type(uint224).max` (2^224^ - 1).
 */
function _maxSupply() internal view virtual returns (uint224) {
   return type(uint224).max;
}

/**
 * @dev Snapshots the totalSupply after it has been increased.
 */
function _mint(address account, uint256 amount) internal virtual override {
   super._mint(account, amount);
   require(totalSupply() <= _maxSupply(), "ERC20Votes: total supply risks overflowing votes");

   _writeCheckpoint(_totalSupplyCheckpoints, _add, amount);
}

/**
 * @dev Snapshots the totalSupply after it has been decreased.
 */
function _burn(address account, uint256 amount) internal virtual override {
   super._burn(account, amount);

   _writeCheckpoint(_totalSupplyCheckpoints, _subtract, amount);
}

/**
 * @dev Move voting power when tokens are transferred.
 *
 * Emits a {DelegateVotesChanged} event.
 */
```

```
function _afterTokenTransfer(
    address from,
    address to,
    uint256 amount
) internal virtual override {
    super._afterTokenTransfer(from, to, amount);

    _moveVotingPower(delegates(from), delegates(to), amount);
}

/**
 * @dev Change delegation for `delegator` to `delegatee`.
 *
 * Emits events {DelegateChanged} and {DelegateVotesChanged}.
 */
function _delegate(address delegator, address delegatee) internal virtual {
    address currentDelegate = delegates(delegator);
    uint256 delegatorBalance = balanceOf(delegator);
    _delegates[delegator] = delegatee;

    emit DelegateChanged(delegator, currentDelegate, delegatee);

    _moveVotingPower(currentDelegate, delegatee, delegatorBalance);
}

function _moveVotingPower(
    address src,
    address dst,
    uint256 amount
) private {
    if (src != dst && amount > 0) {
            if (src != address(0)) {
                (uint256 oldWeight, uint256 newWeight) = _writeCheckpoint(_checkpoints[src], _subtract,
amount);
                emit DelegateVotesChanged(src, oldWeight, newWeight);
            }

            if (dst != address(0)) {
                (uint256 oldWeight, uint256 newWeight) = _writeCheckpoint(_checkpoints[dst], _add,
amount);
                emit DelegateVotesChanged(dst, oldWeight, newWeight);
            }
        }
    }

    function _writeCheckpoint(
        Checkpoint[] storage ckpts,
        function(uint256, uint256) view returns (uint256) op,
        uint256 delta
    ) private returns (uint256 oldWeight, uint256 newWeight) {
        uint256 pos = ckpts.length;
        oldWeight = pos == 0 ? 0 : ckpts[pos - 1].votes;
        newWeight = op(oldWeight, delta);

        if (pos > 0 && ckpts[pos - 1].fromBlock == block.number) {
            ckpts[pos - 1].votes = SafeCast.toUint224(newWeight);
        } else {
            ckpts.push(Checkpoint({fromBlock: SafeCast.toUint32(block.number), votes:
SafeCast.toUint224(newWeight)}));
        }
    }

    function _add(uint256 a, uint256 b) private pure returns (uint256) {
        return a + b;
    }

    function _subtract(uint256 a, uint256 b) private pure returns (uint256) {
        return a - b;
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.6.0) (token/ERC20/extensions/draft-ERC20Permit.sol)

pragma solidity ^0.8.0;

import "./draft-IERC20Permit.sol";
import "../ERC20.sol";
import "../../../utils/cryptography/draft-EIP712.sol";
import "../../../utils/cryptography/ECDSA.sol";
import "../../../utils/Counters.sol";

/**
 * @dev Implementation of the ERC20 Permit extension allowing approvals to be made via
signatures, as defined in
 * https://eips.ethereum.org/EIPS/eip-2612[EIP-2612].
 *
 * Adds the {permit} method, which can be used to change an account's ERC20 allowance (see
{IERC20-allowance}) by
 * presenting a message signed by the account. By not relying on `{IERC20-approve}`, the token
holder account doesn't
 * need to send a transaction, and thus is not required to hold Ether at all.
 *
 * _Available since v3.4._
 */
abstract contract ERC20Permit is ERC20, IERC20Permit, EIP712 {
    using Counters for Counters.Counter;


    mapping(address => Counters.Counter) private _nonces;


    // solhint-disable-next-line var-name-mixedcase
    bytes32 private constant _PERMIT_TYPEHASH =
        keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)");
    /**
     * @dev In previous versions `_PERMIT_TYPEHASH` was declared as `immutable`.
     * However, to ensure consistency with the upgradeable transpiler, we will continue
     * to reserve a slot.
     * @custom:oz-renamed-from _PERMIT_TYPEHASH
     */
    // solhint-disable-next-line var-name-mixedcase
    bytes32 private _PERMIT_TYPEHASH_DEPRECATED_SLOT;


    /**
     * @dev Initializes the {EIP712} domain separator using the `name` parameter, and setting `version` to `"1"`.
     *
     * It's a good idea to use the same `name` that is defined as the ERC20 token name.
     */
    constructor(string memory name) EIP712(name, "1") {}


    /**
     * @dev See {IERC20Permit-permit}.
     */
    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) public virtual override {
        require(block.timestamp <= deadline, "ERC20Permit: expired deadline");

        bytes32 structHash = keccak256(abi.encode(_PERMIT_TYPEHASH, owner, spender, value, _useNonce(owner), deadline));

        bytes32 hash = _hashTypedDataV4(structHash);

        address signer = ECDSA.recover(hash, v, r, s);
        require(signer == owner, "ERC20Permit: invalid signature");

        _approve(owner, spender, value);
    }

    /**
     * @dev See {IERC20Permit-nonces}.
     */
    function nonces(address owner) public view virtual override returns (uint256) {
        return _nonces[owner].current();
    }

    /**
     * @dev See {IERC20Permit-DOMAIN_SEPARATOR}.
     */
    // solhint-disable-next-line func-name-mixedcase
    function DOMAIN_SEPARATOR() external view override returns (bytes32) {
        return _domainSeparatorV4();
    }

    /**
     * @dev "Consume a nonce": return the current value and increment.
     *
     * _Available since v4.1._
     */
    function _useNonce(address owner) internal virtual returns (uint256 current) {
        Counters.Counter storage nonce = _nonces[owner];
        current = nonce.current();
        nonce.increment();
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0) (access/Ownable.sol)

pragma solidity ^0.8.0;

import "../utils/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.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
abstract 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() {
        _transferOwnership(_msgSender());
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        _checkOwner();
        _;
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view virtual returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if the sender is not the owner.
     */
```

```
function _checkOwner() internal view virtual {
    require(owner() == _msgSender(), "Ownable: caller is not the 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 {
    _transferOwnership(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 {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    _transferOwnership(newOwner);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Internal function without access restriction.
 */
function _transferOwnership(address newOwner) internal virtual {
    address oldOwner = _owner;
    _owner = newOwner;
```

```
        emit OwnershipTransferred(oldOwner, newOwner);
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0) (token/ERC20/extensions/ERC20Snapshot.sol)

pragma solidity ^0.8.0;

import "../ERC20.sol";
import "../../../utils/Arrays.sol";
import "../../../utils/Counters.sol";

/**
 * @dev This contract extends an ERC20 token with a snapshot mechanism. When a snapshot is created, the balances and
 * total supply at the time are recorded for later access.
 *
 * This can be used to safely create mechanisms based on token balances such as trustless dividends or weighted voting.
 * In naive implementations it's possible to perform a "double spend" attack by reusing the same balance from different
 * accounts. By using snapshots to calculate dividends or voting power, those attacks no longer apply. It can also be
 * used to create an efficient ERC20 forking mechanism.
 *
 * Snapshots are created by the internal {_snapshot} function, which will emit the {Snapshot} event and return a
 * snapshot id. To get the total supply at the time of a snapshot, call the function {totalSupplyAt} with the snapshot
 * id. To get the balance of an account at the time of a snapshot, call the {balanceOfAt} function with the snapshot id
 * and the account address.
 *
 * NOTE: Snapshot policy can be customized by overriding the {_getCurrentSnapshotId} method. For example, having it
 * return `block.number` will trigger the creation of snapshot at the beginning of each new block. When overriding this
 * function, be careful about the monotonicity of its result. Non-monotonic snapshot ids will break the contract.
 *
 * Implementing snapshots for every block using this method will incur significant gas costs. For a gas-efficient
 * alternative consider {ERC20Votes}.
 *
 * ==== Gas Costs
 *
 * Snapshots are efficient. Snapshot creation is _O(1)_. Retrieval of balances or total supply from a snapshot is _O(log
 * n)_ in the number of snapshots that have been created, although _n_ for a specific account will generally be much
 * smaller since identical balances in subsequent snapshots are stored as a single entry.
 *
 * There is a constant overhead for normal ERC20 transfers due to the additional snapshot bookkeeping. This overhead is
 * only significant for the first transfer that immediately follows a snapshot for a particular account. Subsequent
 * transfers will have normal cost until the next snapshot, and so on.
 */

abstract contract ERC20Snapshot is ERC20 {
    // Inspired by Jordi Baylina's MiniMeToken to record historical balances:
    // https://github.com/Giveth/minime/blob/ea04d950eea153a04c51fa510b068b9dded390cb/contracts/MiniMeToken.sol

    using Arrays for uint256[];
    using Counters for Counters.Counter;
    // Snapshotted values have arrays of ids and the value corresponding to that id. These could be an array of a
    // Snapshot struct, but that would impede usage of functions that work on an array.
    struct Snapshots {
        uint256[] ids;
        uint256[] values;
    }

    mapping(address => Snapshots) private _accountBalanceSnapshots;
    Snapshots private _totalSupplySnapshots;

    // Snapshot ids increase monotonically, with the first value being 1. An id of 0 is invalid.
    Counters.Counter private _currentSnapshotId;

    /**
     * @dev Emitted by {_snapshot} when a snapshot identified by `id` is created.
     */
    event Snapshot(uint256 id);

    /**
     * @dev Creates a new snapshot and returns its snapshot id.
     *
     * Emits a {Snapshot} event that contains the same id.
     *
     * {_snapshot} is `internal` and you have to decide how to expose it externally. Its usage may be restricted to a
     * set of accounts, for example using {AccessControl}, or it may be open to the public.
     *
     * [WARNING]
     * ====
     * While an open way of calling {_snapshot} is required for certain trust minimization mechanisms such as forking,
     * you must consider that it can potentially be used by attackers in two ways.
     *
     * First, it can be used to increase the cost of retrieval of values from snapshots, although it will
grow
     * logarithmically thus rendering this attack ineffective in the long term. Second, it can be used to
target
     * specific accounts and increase the cost of ERC20 transfers for them, in the ways specified in the
Gas Costs
     * section above.
     *
     * We haven't measured the actual numbers; if this is something you're interested in please reach
out to us.
     * ====
     */
    function _snapshot() internal virtual returns (uint256) {
        _currentSnapshotId.increment();

        uint256 currentId = _getCurrentSnapshotId();
        emit Snapshot(currentId);
        return currentId;
    }

    /**
     * @dev Get the current snapshotId
     */
    function _getCurrentSnapshotId() internal view virtual returns (uint256) {
        return _currentSnapshotId.current();
    }

    /**
     * @dev Retrieves the balance of `account` at the time `snapshotId` was created.
     */
    function balanceOfAt(address account, uint256 snapshotId) public view virtual returns (uint256) {
        (bool snapshotted, uint256 value) = _valueAt(snapshotId, _accountBalanceSnapshots[account]);

        return snapshotted ? value : balanceOf(account);
    }

    /**
     * @dev Retrieves the total supply at the time `snapshotId` was created.
     */
    function totalSupplyAt(uint256 snapshotId) public view virtual returns (uint256) {
        (bool snapshotted, uint256 value) = _valueAt(snapshotId, _totalSupplySnapshots);

        return snapshotted ? value : totalSupply();
    }

    // Update balance and/or total supply snapshots before the values are modified. This is
implemented
    // in the _beforeTokenTransfer hook, which is executed for _mint, _burn, and _transfer
operations.
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual override {
        super._beforeTokenTransfer(from, to, amount);

        if (from == address(0)) {
            // mint
            _updateAccountSnapshot(to);
            _updateTotalSupplySnapshot();
        } else if (to == address(0)) {
            // burn
            _updateAccountSnapshot(from);
            _updateTotalSupplySnapshot();
        } else {
            // transfer
            _updateAccountSnapshot(from);
            _updateAccountSnapshot(to);
        }
    }

    function _valueAt(uint256 snapshotId, Snapshots storage snapshots) private view returns (bool, uint256) {
        require(snapshotId > 0, "ERC20Snapshot: id is 0");
        require(snapshotId <= _getCurrentSnapshotId(), "ERC20Snapshot: nonexistent id");

        // When a valid snapshot is queried, there are three possibilities:
        //  a) The queried value was not modified after the snapshot was taken. Therefore, a snapshot entry was never
        //  created for this id, and all stored snapshot ids are smaller than the requested one. The value that corresponds
        //  to this id is the current one.
        //  b) The queried value was modified after the snapshot was taken. Therefore, there will be an entry with the
        //  requested id, and its value is the one to return.
        //  c) More snapshots were created after the requested one, and the queried value was later modified. There will be
        //  no entry for the requested id: the value that corresponds to it is that of the smallest snapshot id that is
        //  larger than the requested one.
        //
        // In summary, we need to find an element in an array, returning the index of the smallest value that is larger if
        // it is not found, unless said value doesn't exist (e.g. when all values are smaller). Arrays.findUpperBound does
        // exactly this.
        uint256 index = snapshots.ids.findUpperBound(snapshotId);

        if (index == snapshots.ids.length) {
            return (false, 0);
        } else {
            return (true, snapshots.values[index]);
        }
    }

    function _updateAccountSnapshot(address account) private {
        _updateSnapshot(_accountBalanceSnapshots[account], balanceOf(account));
    }

    function _updateTotalSupplySnapshot() private {
        _updateSnapshot(_totalSupplySnapshots, totalSupply());
    }

    function _updateSnapshot(Snapshots storage snapshots, uint256 currentValue) private {
        uint256 currentId = _getCurrentSnapshotId();
        if (_lastSnapshotId(snapshots.ids) < currentId) {
            snapshots.ids.push(currentId);
            snapshots.values.push(currentValue);
        }
    }

    function _lastSnapshotId(uint256[] storage ids) private view returns (uint256) {
        if (ids.length == 0) {
            return 0;
        } else {
            return ids[ids.length - 1];
        }
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.5.0) (token/ERC20/extensions/ERC20Burnable.sol)

pragma solidity ^0.8.0;

import "../ERC20.sol";
import "../../../utils/Context.sol";

/**
 * @dev Extension of {ERC20} that allows token holders to destroy both their own
 * tokens and those that they have an allowance for, in a way that can be
 * recognized off-chain (via event analysis).
 */
abstract contract ERC20Burnable is Context, ERC20 {
    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See {ERC20-_burn}.
     */
    function burn(uint256 amount) public virtual {
        _burn(_msgSender(), amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, deducting from the caller's
     * allowance.
     *
     * See {ERC20-_burn} and {ERC20-allowance}.
```

```
   *
   * Requirements:
   *
   * - the caller must have allowance for ``accounts``'s tokens of at least
   * `amount`.
   */
  function burnFrom(address account, uint256 amount) public virtual {
    _spendAllowance(account, _msgSender(), amount);
    _burn(account, amount);
  }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0) (token/ERC20/ERC20.sol)

pragma solidity ^0.8.0;

import "./IERC20.sol";
import "./extensions/IERC20Metadata.sol";
import "../../utils/Context.sol";

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin Contracts guidelines: functions revert
 * instead returning `false` on failure. This behavior is nonetheless
 * conventional and does not conflict with the expectations of ERC20
 * applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20, IERC20Metadata {
    mapping(address => uint256) private _balances;

    mapping(address => mapping(address => uint256)) private _allowances;

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    /**
     * @dev Sets the values for {name} and {symbol}.
     *
     * The default value of {decimals} is 18. To select a different value for
     * {decimals} you should overload it.
     *
     * All two of these values are immutable: they can only be set once during
     * construction.
     */
    constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

    /**
     * @dev Returns the name of the token.
     */
    function name() public view virtual override returns (string memory) {
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view virtual override returns (string memory) {
        return _symbol;
    }

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5.05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei. This is the value {ERC20} uses, unless this function is
     * overridden;
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view virtual override returns (uint8) {
        return 18;
    }

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

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

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `to` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address to, uint256 amount) public virtual override returns (bool) {
        address owner = _msgSender();
        _transfer(owner, to, amount);
        return true;
    }

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

    /**
     * @dev See {IERC20-approve}.
     *
     * NOTE: If `amount` is the maximum `uint256`, the allowance is not updated on
     * `transferFrom`. This is semantically equivalent to an infinite approval.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        address owner = _msgSender();
        _approve(owner, spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
```

```
 *
 * Emits an {Approval} event indicating the updated allowance. This is not
 * required by the EIP. See the note at the beginning of {ERC20}.
 *
 * NOTE: Does not update the allowance if the current allowance
 * is the maximum `uint256`.
 *
 * Requirements:
 *
 * - `from` and `to` cannot be the zero address.
 * - `from` must have a balance of at least `amount`.
 * - the caller must have allowance for ``from``'s tokens of at least
 * `amount`.
 */
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
    address spender = _msgSender();
    _spendAllowance(from, spender, amount);
    _transfer(from, to, amount);
    return true;
}

/**
 * @dev Atomically increases the allowance granted to `spender` by the caller.
 *
 * This is an alternative to {approve} that can be used as a mitigation for
 * problems described in {IERC20-approve}.
 *
```

```
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        address owner = _msgSender();
        _approve(owner, spender, allowance(owner, spender) + addedValue);
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        address owner = _msgSender();
        uint256 currentAllowance = allowance(owner, spender);
        require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below zero");
        unchecked {
            _approve(owner, spender, currentAllowance - subtractedValue);
        }

        return true;
    }

    /**
     * @dev Moves `amount` of tokens from `from` to `to`.
     *
     * This internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `from` must have a balance of at least `amount`.
     */
    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);
        uint256 fromBalance = _balances[from];
        require(fromBalance >= amount, "ERC20: transfer amount exceeds balance");
        unchecked {
            _balances[from] = fromBalance - amount;
        }
        _balances[to] += amount;

        emit Transfer(from, to, amount);

        _afterTokenTransfer(from, to, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");

        _beforeTokenTransfer(address(0), account, amount);

        _totalSupply += amount;
        _balances[account] += amount;
        emit Transfer(address(0), account, amount);

        _afterTokenTransfer(address(0), account, amount);
```

```
}

/**
 * @dev Destroys `amount` tokens from `account`, reducing the
 * total supply.
 *
 * Emits a {Transfer} event with `to` set to the zero address.
 *
 * Requirements:
 *
 * - `account` cannot be the zero address.
 * - `account` must have at least `amount` tokens.
 */
function _burn(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: burn from the zero address");

    _beforeTokenTransfer(account, address(0), amount);

    uint256 accountBalance = _balances[account];
    require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
    unchecked {
        _balances[account] = accountBalance - amount;
    }
    _totalSupply -= amount;

    emit Transfer(account, address(0), amount);

    _afterTokenTransfer(account, address(0), amount);
}

/**
```

```
 * @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.
 *
 * This internal function is equivalent to `approve`, and can be used to
 * e.g. set automatic allowances for certain subsystems, etc.
 *
 * Emits an {Approval} event.
 *
 * Requirements:
 *
 * - `owner` cannot be the zero address.
 * - `spender` cannot be the zero address.
 */
function _approve(
    address owner,
    address spender,
    uint256 amount
) internal virtual {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}

/**
 * @dev Updates `owner` s allowance for `spender` based on spent `amount`.
 *
 * Does not update the allowance amount in case of infinite allowance.
 * Revert if not enough allowance is available.
 *
 * Might emit an {Approval} event.
```

```
     */
    function _spendAllowance(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        uint256 currentAllowance = allowance(owner, spender);
        if (currentAllowance != type(uint256).max) {
            require(currentAllowance >= amount, "ERC20: insufficient allowance");
            unchecked {
                _approve(owner, spender, currentAllowance - amount);
            }
        }
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using
Hooks].
     */
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}

    /**
     * @dev Hook that is called after any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * has been transferred to `to`.
     * - when `from` is zero, `amount` tokens have been minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens have been burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using
Hooks].
     */
    function _afterTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (utils/Counters.sol)

pragma solidity ^0.8.0;
```

```
/**
 * @title Counters
 * @author Matt Condon (@shrugs)
 * @dev Provides counters that can only be incremented, decremented or reset. This can be used e.g. to track the number
 * of elements in a mapping, issuing ERC721 ids, or counting request ids.
 *
 * Include with `using Counters for Counters.Counter;`
 */
library Counters {
    struct Counter {
        // This variable should never be directly accessed by users of the library: interactions must be restricted to
        // the library's function. As of Solidity v0.5.2, this cannot be enforced, though there is a proposal to add
        // this feature: see https://github.com/ethereum/solidity/issues/4637
        uint256 _value; // default: 0
    }

    function current(Counter storage counter) internal view returns (uint256) {
        return counter._value;
    }

    function increment(Counter storage counter) internal {
        unchecked {
            counter._value += 1;
        }
    }

    function decrement(Counter storage counter) internal {
        uint256 value = counter._value;
        require(value > 0, "Counter: decrement overflow");
        unchecked {
            counter._value = value - 1;
        }
    }

    function reset(Counter storage counter) internal {
        counter._value = 0;
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.3) (utils/cryptography/ECDSA.sol)

pragma solidity ^0.8.0;

import "../Strings.sol";

/**
 * @dev Elliptic Curve Digital Signature Algorithm (ECDSA) operations.
 *
 * These functions can be used to verify that a message was signed by the holder
 * of the private keys of a given address.
 */
library ECDSA {
    enum RecoverError {
        NoError,
        InvalidSignature,
        InvalidSignatureLength,
        InvalidSignatureS,
        InvalidSignatureV
    }
```

```
function _throwError(RecoverError error) private pure {
    if (error == RecoverError.NoError) {
        return; // no error: do nothing
    } else if (error == RecoverError.InvalidSignature) {
        revert("ECDSA: invalid signature");
    } else if (error == RecoverError.InvalidSignatureLength) {
        revert("ECDSA: invalid signature length");
    } else if (error == RecoverError.InvalidSignatureS) {
        revert("ECDSA: invalid signature 's' value");
    } else if (error == RecoverError.InvalidSignatureV) {
        revert("ECDSA: invalid signature 'v' value");
    }
}

/**
 * @dev Returns the address that signed a hashed message (`hash`) with
 * `signature` or error string. This address can then be used for verification purposes.
 *
 * The `ecrecover` EVM opcode allows for malleable (non-unique) signatures:
 * this function rejects them by requiring the `s` value to be in the lower
 * half order, and the `v` value to be either 27 or 28.
 *
 * IMPORTANT: `hash` _must_ be the result of a hash operation for the
 * verification to be secure: it is possible to craft signatures that
 * recover to arbitrary addresses for non-hashed data. A safe way to ensure
 * this is by receiving a hash of the original message (which may otherwise
 * be too long), and then calling {toEthSignedMessageHash} on it.
 *
 * Documentation for signature generation:
 * - with https://web3js.readthedocs.io/en/v1.3.4/web3-eth-accounts.html#sign[Web3.js]
 * - with https://docs.ethers.io/v5/api/signer/#Signer-signMessage[ethers]
```

```
     *
     * _Available since v4.3._
     */
    function tryRecover(bytes32 hash, bytes memory signature) internal pure returns (address,
RecoverError) {
        if (signature.length == 65) {
            bytes32 r;
            bytes32 s;
            uint8 v;
            // ecrecover takes the signature parameters, and the only way to get them
            // currently is to use assembly.
            /// @solidity memory-safe-assembly
            assembly {
                r := mload(add(signature, 0x20))
                s := mload(add(signature, 0x40))
                v := byte(0, mload(add(signature, 0x60)))
            }
            return tryRecover(hash, v, r, s);
        } else {
            return (address(0), RecoverError.InvalidSignatureLength);
        }
    }

    /**
     * @dev Returns the address that signed a hashed message (`hash`) with
     * `signature`. This address can then be used for verification purposes.
     *
     * The `ecrecover` EVM opcode allows for malleable (non-unique) signatures:
     * this function rejects them by requiring the `s` value to be in the lower
     * half order, and the `v` value to be either 27 or 28.
     *
     * IMPORTANT: `hash` _must_ be the result of a hash operation for the
     * verification to be secure: it is possible to craft signatures that
     * recover to arbitrary addresses for non-hashed data. A safe way to ensure
     * this is by receiving a hash of the original message (which may otherwise
     * be too long), and then calling {toEthSignedMessageHash} on it.
     */
    function recover(bytes32 hash, bytes memory signature) internal pure returns (address) {
        (address recovered, RecoverError error) = tryRecover(hash, signature);
        _throwError(error);
        return recovered;
    }

    /**
     * @dev Overload of {ECDSA-tryRecover} that receives the `r` and `vs` short-signature fields separately.
     *
     * See https://eips.ethereum.org/EIPS/eip-2098[EIP-2098 short signatures]
     *
     * _Available since v4.3._
     */
    function tryRecover(
        bytes32 hash,
        bytes32 r,
        bytes32 vs
    ) internal pure returns (address, RecoverError) {
        bytes32 s = vs & bytes32(0x7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff);
        uint8 v = uint8((uint256(vs) >> 255) + 27);
        return tryRecover(hash, v, r, s);
    }

    /**
```

```
     * @dev Overload of {ECDSA-recover} that receives the `r and `vs` short-signature fields separately.
     *
     * _Available since v4.2._
     */
    function recover(
        bytes32 hash,
        bytes32 r,
        bytes32 vs
    ) internal pure returns (address) {
        (address recovered, RecoverError error) = tryRecover(hash, r, vs);
        _throwError(error);
        return recovered;
    }

    /**
     * @dev Overload of {ECDSA-tryRecover} that receives the `v`,
     * `r` and `s` signature fields separately.
     *
     * _Available since v4.3._
     */
    function tryRecover(
        bytes32 hash,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) internal pure returns (address, RecoverError) {
        // EIP-2 still allows signature malleability for ecrecover(). Remove this possibility and make the signature
        // unique. Appendix F in the Ethereum Yellow paper (https://ethereum.github.io/yellowpaper/paper.pdf), defines
        // the valid range for s in (301): 0 < s < secp256k1n ÷ 2 + 1, and for v in (302): v ∈ {27, 28}. Most
        // signatures from current libraries generate a unique signature with an s-value in the lower half order.
        //
        // If your library generates malleable signatures, such as s-values in the upper range, calculate a new s-value
        // with 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141 - s1 and flip v from 27 to 28 or
        // vice versa. If your library also generates signatures with 0/1 for v instead 27/28, add 27 to v to accept
        // these malleable signatures as well.
        if (uint256(s) > 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0) {
            return (address(0), RecoverError.InvalidSignatureS);
        }
        if (v != 27 && v != 28) {
            return (address(0), RecoverError.InvalidSignatureV);
        }

        // If the signature is valid (and not malleable), return the signer address
        address signer = ecrecover(hash, v, r, s);
        if (signer == address(0)) {
            return (address(0), RecoverError.InvalidSignature);
        }

        return (signer, RecoverError.NoError);
    }

    /**
     * @dev Overload of {ECDSA-recover} that receives the `v`,
     * `r` and `s` signature fields separately.
     */
    function recover(
        bytes32 hash,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) internal pure returns (address) {
        (address recovered, RecoverError error) = tryRecover(hash, v, r, s);
        _throwError(error);
        return recovered;
    }

    /**
     * @dev Returns an Ethereum Signed Message, created from a `hash`. This
     * produces hash corresponding to the one signed with the
     * https://eth.wiki/json-rpc/API#eth_sign[`eth_sign`]
     * JSON-RPC method as part of EIP-191.
     *
     * See {recover}.
     */
    function toEthSignedMessageHash(bytes32 hash) internal pure returns (bytes32) {
        // 32 is the length in bytes of hash,
        // enforced by the type signature above
        return keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", hash));
    }

    /**
     * @dev Returns an Ethereum Signed Message, created from `s`. This
     * produces hash corresponding to the one signed with the
     * https://eth.wiki/json-rpc/API#eth_sign[`eth_sign`]
     * JSON-RPC method as part of EIP-191.
     *
     * See {recover}.
     */
```

```
   function toEthSignedMessageHash(bytes memory s) internal pure returns (bytes32) {
       return keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n",
Strings.toString(s.length), s));
   }


   /**
    * @dev Returns an Ethereum Signed Typed Data, created from a
    * `domainSeparator` and a `structHash`. This produces hash corresponding
    * to the one signed with the
    * https://eips.ethereum.org/EIPS/eip-712[`eth_signTypedData`]
    * JSON-RPC method as part of EIP-712.
    *
    * See {recover}.
    */
   function toTypedDataHash(bytes32 domainSeparator, bytes32 structHash) internal pure returns
(bytes32) {
       return keccak256(abi.encodePacked("\x19\x01", domainSeparator, structHash));
   }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (utils/cryptography/draft-EIP712.sol)


pragma solidity ^0.8.0;


import "./ECDSA.sol";


/**
 * @dev https://eips.ethereum.org/EIPS/eip-712[EIP 712] is a standard for hashing and signing of
typed structured data.
 *
 * The encoding specified in the EIP is very generic, and such a generic implementation in Solidity is
not feasible,
 * thus this contract does not implement the encoding itself. Protocols need to implement the type-specific encoding
 * they need in their contracts using a combination of `abi.encode` and `keccak256`.
 *
 * This contract implements the EIP 712 domain separator ({_domainSeparatorV4}) that is used as part of the encoding
 * scheme, and the final step of the encoding to obtain the message digest that is then signed via ECDSA
 * ({_hashTypedDataV4}).
 *
 * The implementation of the domain separator was designed to be as efficient as possible while still properly updating
 * the chain id to protect against replay attacks on an eventual fork of the chain.
 *
 * NOTE: This contract implements the version of the encoding known as "v4", as implemented by the JSON RPC method
 * https://docs.metamask.io/guide/signing-data.html[`eth_signTypedDataV4` in MetaMask].
 *
 * _Available since v3.4._
 */
abstract contract EIP712 {
   /* solhint-disable var-name-mixedcase */
   // Cache the domain separator as an immutable value, but also store the chain id that it corresponds to, in order to
   // invalidate the cached domain separator if the chain id changes.
   bytes32 private immutable _CACHED_DOMAIN_SEPARATOR;
   uint256 private immutable _CACHED_CHAIN_ID;
   address private immutable _CACHED_THIS;

   bytes32 private immutable _HASHED_NAME;
   bytes32 private immutable _HASHED_VERSION;
   bytes32 private immutable _TYPE_HASH;
```

```
    /* solhint-enable var-name-mixedcase */

    /**
     * @dev Initializes the domain separator and parameter caches.
     *
     * The meaning of `name` and `version` is specified in
     * https://eips.ethereum.org/EIPS/eip-712#definition-of-domainseparator[EIP 712]:
     *
     * - `name`: the user readable name of the signing domain, i.e. the name of the DApp or the protocol.
     * - `version`: the current major version of the signing domain.
     *
     * NOTE: These parameters cannot be changed except through a xref:learn::upgrading-smart-contracts.adoc[smart
     * contract upgrade].
     */
    constructor(string memory name, string memory version) {
        bytes32 hashedName = keccak256(bytes(name));
        bytes32 hashedVersion = keccak256(bytes(version));
        bytes32 typeHash = keccak256(
            "EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"
        );
        _HASHED_NAME = hashedName;
        _HASHED_VERSION = hashedVersion;
        _CACHED_CHAIN_ID = block.chainid;
        _CACHED_DOMAIN_SEPARATOR = _buildDomainSeparator(typeHash, hashedName, hashedVersion);
        _CACHED_THIS = address(this);
        _TYPE_HASH = typeHash;
    }

    /**
```

```
     * @dev Returns the domain separator for the current chain.
     */
    function _domainSeparatorV4() internal view returns (bytes32) {
        if (address(this) == _CACHED_THIS && block.chainid == _CACHED_CHAIN_ID) {
            return _CACHED_DOMAIN_SEPARATOR;
        } else {
            return _buildDomainSeparator(_TYPE_HASH, _HASHED_NAME, _HASHED_VERSION);
        }
    }

    function _buildDomainSeparator(
        bytes32 typeHash,
        bytes32 nameHash,
        bytes32 versionHash
    ) private view returns (bytes32) {
        return keccak256(abi.encode(typeHash, nameHash, versionHash, block.chainid, address(this)));
    }

    /**
     * @dev Given an already https://eips.ethereum.org/EIPS/eip-712#definition-of-hashstruct[hashed struct], this
     * function returns the hash of the fully encoded EIP712 message for this domain.
     *
     * This hash can be used together with {ECDSA-recover} to obtain the signer of a message. For example:
     *
     * ```solidity
     * bytes32 digest = _hashTypedDataV4(keccak256(abi.encode(
     *     keccak256("Mail(address to,string contents)"),
     *     mailTo,
     *     keccak256(bytes(mailContents))
     * )));
```

```
     * address signer = ECDSA.recover(digest, signature);
     * ```
     */
    function _hashTypedDataV4(bytes32 structHash) internal view virtual returns (bytes32) {
        return ECDSA.toTypedDataHash(_domainSeparatorV4(), structHash);
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (token/ERC20/extensions/draft-IERC20Permit.sol)

pragma solidity ^0.8.0;

/**
 * @dev Interface of the ERC20 Permit extension allowing approvals to be made via signatures, as defined in
 * https://eips.ethereum.org/EIPS/eip-2612[EIP-2612].
 *
 * Adds the {permit} method, which can be used to change an account's ERC20 allowance (see {IERC20-allowance}) by
 * presenting a message signed by the account. By not relying on {IERC20-approve}, the token holder account doesn't
 * need to send a transaction, and thus is not required to hold Ether at all.
 */
interface IERC20Permit {
    /**
     * @dev Sets `value` as the allowance of `spender` over ``owner``'s tokens,
     * given ``owner``'s signed approval.
     *
     * IMPORTANT: The same issues {IERC20-approve} has related to transaction
     * ordering also apply here.
     *
     * Emits an {Approval} event.
```

```
 *
 * Requirements:
 *
 * - `spender` cannot be the zero address.
 * - `deadline` must be a timestamp in the future.
 * - `v`, `r` and `s` must be a valid `secp256k1` signature from `owner`
 * over the EIP712-formatted function arguments.
 * - the signature must use ``owner``'s current nonce (see {nonces}).
 *
 * For more information on the signature format, see the
 * https://eips.ethereum.org/EIPS/eip-2612#specification[relevant EIP
 * section].
 */
function permit(
    address owner,
    address spender,
    uint256 value,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external;

/**
 * @dev Returns the current nonce for `owner`. This value must be
 * included whenever a signature is generated for {permit}.
 *
 * Every successful call to {permit} increases ``owner``'s nonce by one. This
 * prevents a signature from being used multiple times.
 */
function nonces(address owner) external view returns (uint256);
```

```
    /**
     * @dev Returns the domain separator used in the encoding of the signature for {permit}, as defined by {EIP712}.
     */
    // solhint-disable-next-line func-name-mixedcase
    function DOMAIN_SEPARATOR() external view returns (bytes32);
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0) (utils/math/SafeCast.sol)

pragma solidity ^0.8.0;

/**
 * @dev Wrappers over Solidity's uintXX/intXX casting operators with added overflow
 * checks.
 *
 * Downcasting from uint256/int256 in Solidity does not revert on overflow. This can
 * easily result in undesired exploitation or bugs, since developers usually
 * assume that overflows raise errors. `SafeCast` restores this intuition by
 * reverting the transaction when such an operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 *
 * Can be combined with {SafeMath} and {SignedSafeMath} to extend it to smaller types, by performing
 * all math on `uint256` and `int256` and then downcasting.
 */
library SafeCast {
    /**
     * @dev Returns the downcasted uint248 from uint256, reverting on
 * overflow (when the input is greater than largest uint248).
 *
 * Counterpart to Solidity's `uint248` operator.
 *
 * Requirements:
 *
 * - input must fit into 248 bits
 *
 * _Available since v4.7._
 */
function toUint248(uint256 value) internal pure returns (uint248) {
    require(value <= type(uint248).max, "SafeCast: value doesn't fit in 248 bits");
    return uint248(value);
}

/**
 * @dev Returns the downcasted uint240 from uint256, reverting on
 * overflow (when the input is greater than largest uint240).
 *
 * Counterpart to Solidity's `uint240` operator.
 *
 * Requirements:
 *
 * - input must fit into 240 bits
 *
 * _Available since v4.7._
 */
function toUint240(uint256 value) internal pure returns (uint240) {
    require(value <= type(uint240).max, "SafeCast: value doesn't fit in 240 bits");
    return uint240(value);
}
```

```
/**
 * @dev Returns the downcasted uint232 from uint256, reverting on
 * overflow (when the input is greater than largest uint232).
 *
 * Counterpart to Solidity's `uint232` operator.
 *
 * Requirements:
 *
 * - input must fit into 232 bits
 *
 * _Available since v4.7._
 */
function toUint232(uint256 value) internal pure returns (uint232) {
    require(value <= type(uint232).max, "SafeCast: value doesn't fit in 232 bits");
    return uint232(value);
}

/**
 * @dev Returns the downcasted uint224 from uint256, reverting on
 * overflow (when the input is greater than largest uint224).
 *
 * Counterpart to Solidity's `uint224` operator.
 *
 * Requirements:
 *
 * - input must fit into 224 bits
 *
 * _Available since v4.2._
 */
function toUint224(uint256 value) internal pure returns (uint224) {
    require(value <= type(uint224).max, "SafeCast: value doesn't fit in 224 bits");
    return uint224(value);
}

/**
 * @dev Returns the downcasted uint216 from uint256, reverting on
 * overflow (when the input is greater than largest uint216).
 *
 * Counterpart to Solidity's `uint216` operator.
 *
 * Requirements:
 *
 * - input must fit into 216 bits
 *
 * _Available since v4.7._
 */
function toUint216(uint256 value) internal pure returns (uint216) {
    require(value <= type(uint216).max, "SafeCast: value doesn't fit in 216 bits");
    return uint216(value);
}

/**
 * @dev Returns the downcasted uint208 from uint256, reverting on
 * overflow (when the input is greater than largest uint208).
 *
 * Counterpart to Solidity's `uint208` operator.
 *
 * Requirements:
 *
 * - input must fit into 208 bits
 *
 * _Available since v4.7._
 */
function toUint208(uint256 value) internal pure returns (uint208) {
   require(value <= type(uint208).max, "SafeCast: value doesn't fit in 208 bits");
   return uint208(value);
}

/**
 * @dev Returns the downcasted uint200 from uint256, reverting on
 * overflow (when the input is greater than largest uint200).
 *
 * Counterpart to Solidity's `uint200` operator.
 *
 * Requirements:
 *
 * - input must fit into 200 bits
 *
 * _Available since v4.7._
 */
function toUint200(uint256 value) internal pure returns (uint200) {
   require(value <= type(uint200).max, "SafeCast: value doesn't fit in 200 bits");
   return uint200(value);
}

/**
 * @dev Returns the downcasted uint192 from uint256, reverting on
 * overflow (when the input is greater than largest uint192).
 *
 * Counterpart to Solidity's `uint192` operator.
 *
 * Requirements:
```

```
 *
 * - input must fit into 192 bits
 *
 * _Available since v4.7._
 */
function toUint192(uint256 value) internal pure returns (uint192) {
   require(value <= type(uint192).max, "SafeCast: value doesn't fit in 192 bits");
   return uint192(value);
}

/**
 * @dev Returns the downcasted uint184 from uint256, reverting on
 * overflow (when the input is greater than largest uint184).
 *
 * Counterpart to Solidity's `uint184` operator.
 *
 * Requirements:
 *
 * - input must fit into 184 bits
 *
 * _Available since v4.7._
 */
function toUint184(uint256 value) internal pure returns (uint184) {
   require(value <= type(uint184).max, "SafeCast: value doesn't fit in 184 bits");
   return uint184(value);
}

/**
 * @dev Returns the downcasted uint176 from uint256, reverting on
 * overflow (when the input is greater than largest uint176).
 *
 * Counterpart to Solidity's `uint176` operator.
 *
 * Requirements:
 *
 * - input must fit into 176 bits
 *
 * _Available since v4.7._
 */
function toUint176(uint256 value) internal pure returns (uint176) {
    require(value <= type(uint176).max, "SafeCast: value doesn't fit in 176 bits");
    return uint176(value);
}

/**
 * @dev Returns the downcasted uint168 from uint256, reverting on
 * overflow (when the input is greater than largest uint168).
 *
 * Counterpart to Solidity's `uint168` operator.
 *
 * Requirements:
 *
 * - input must fit into 168 bits
 *
 * _Available since v4.7._
 */
function toUint168(uint256 value) internal pure returns (uint168) {
    require(value <= type(uint168).max, "SafeCast: value doesn't fit in 168 bits");
    return uint168(value);
}

/**
```

```
 * @dev Returns the downcasted uint160 from uint256, reverting on
 * overflow (when the input is greater than largest uint160).
 *
 * Counterpart to Solidity's `uint160` operator.
 *
 * Requirements:
 *
 * - input must fit into 160 bits
 *
 * _Available since v4.7._
 */
function toUint160(uint256 value) internal pure returns (uint160) {
   require(value <= type(uint160).max, "SafeCast: value doesn't fit in 160 bits");
   return uint160(value);
}

/**
 * @dev Returns the downcasted uint152 from uint256, reverting on
 * overflow (when the input is greater than largest uint152).
 *
 * Counterpart to Solidity's `uint152` operator.
 *
 * Requirements:
 *
 * - input must fit into 152 bits
 *
 * _Available since v4.7._
 */
function toUint152(uint256 value) internal pure returns (uint152) {
   require(value <= type(uint152).max, "SafeCast: value doesn't fit in 152 bits");
   return uint152(value);
}

/**
 * @dev Returns the downcasted uint144 from uint256, reverting on
 * overflow (when the input is greater than largest uint144).
 *
 * Counterpart to Solidity's `uint144` operator.
 *
 * Requirements:
 *
 * - input must fit into 144 bits
 *
 * _Available since v4.7._
 */
function toUint144(uint256 value) internal pure returns (uint144) {
    require(value <= type(uint144).max, "SafeCast: value doesn't fit in 144 bits");
    return uint144(value);
}

/**
 * @dev Returns the downcasted uint136 from uint256, reverting on
 * overflow (when the input is greater than largest uint136).
 *
 * Counterpart to Solidity's `uint136` operator.
 *
 * Requirements:
 *
 * - input must fit into 136 bits
 *
 * _Available since v4.7._
 */
```

```
function toUint136(uint256 value) internal pure returns (uint136) {
    require(value <= type(uint136).max, "SafeCast: value doesn't fit in 136 bits");
    return uint136(value);
}

/**
 * @dev Returns the downcasted uint128 from uint256, reverting on
 * overflow (when the input is greater than largest uint128).
 *
 * Counterpart to Solidity's `uint128` operator.
 *
 * Requirements:
 *
 * - input must fit into 128 bits
 *
 * _Available since v2.5._
 */
function toUint128(uint256 value) internal pure returns (uint128) {
    require(value <= type(uint128).max, "SafeCast: value doesn't fit in 128 bits");
    return uint128(value);
}

/**
 * @dev Returns the downcasted uint120 from uint256, reverting on
 * overflow (when the input is greater than largest uint120).
 *
 * Counterpart to Solidity's `uint120` operator.
 *
 * Requirements:
 *
 * - input must fit into 120 bits
 *
 * _Available since v4.7._
 */
function toUint120(uint256 value) internal pure returns (uint120) {
    require(value <= type(uint120).max, "SafeCast: value doesn't fit in 120 bits");
    return uint120(value);
}

/**
 * @dev Returns the downcasted uint112 from uint256, reverting on
 * overflow (when the input is greater than largest uint112).
 *
 * Counterpart to Solidity's `uint112` operator.
 *
 * Requirements:
 *
 * - input must fit into 112 bits
 *
 * _Available since v4.7._
 */
function toUint112(uint256 value) internal pure returns (uint112) {
    require(value <= type(uint112).max, "SafeCast: value doesn't fit in 112 bits");
    return uint112(value);
}

/**
 * @dev Returns the downcasted uint104 from uint256, reverting on
 * overflow (when the input is greater than largest uint104).
 *
 * Counterpart to Solidity's `uint104` operator.
 *
 * Requirements:
 *
 * - input must fit into 104 bits
 *
 * _Available since v4.7._
 */
function toUint104(uint256 value) internal pure returns (uint104) {
    require(value <= type(uint104).max, "SafeCast: value doesn't fit in 104 bits");
    return uint104(value);
}

/**
 * @dev Returns the downcasted uint96 from uint256, reverting on
 * overflow (when the input is greater than largest uint96).
 *
 * Counterpart to Solidity's `uint96` operator.
 *
 * Requirements:
 *
 * - input must fit into 96 bits
 *
 * _Available since v4.2._
 */
function toUint96(uint256 value) internal pure returns (uint96) {
    require(value <= type(uint96).max, "SafeCast: value doesn't fit in 96 bits");
    return uint96(value);
}

/**
 * @dev Returns the downcasted uint88 from uint256, reverting on
 * overflow (when the input is greater than largest uint88).
 *
 * Counterpart to Solidity's `uint88` operator.
 *
 * Requirements:
 *
 * - input must fit into 88 bits
 *
 * _Available since v4.7._
 */
function toUint88(uint256 value) internal pure returns (uint88) {
    require(value <= type(uint88).max, "SafeCast: value doesn't fit in 88 bits");
    return uint88(value);
}

/**
 * @dev Returns the downcasted uint80 from uint256, reverting on
 * overflow (when the input is greater than largest uint80).
 *
 * Counterpart to Solidity's `uint80` operator.
 *
 * Requirements:
 *
 * - input must fit into 80 bits
 *
 * _Available since v4.7._
 */
function toUint80(uint256 value) internal pure returns (uint80) {
    require(value <= type(uint80).max, "SafeCast: value doesn't fit in 80 bits");
    return uint80(value);
}
```

```
/**
 * @dev Returns the downcasted uint72 from uint256, reverting on
 * overflow (when the input is greater than largest uint72).
 *
 * Counterpart to Solidity's `uint72` operator.
 *
 * Requirements:
 *
 * - input must fit into 72 bits
 *
 * _Available since v4.7._
 */
function toUint72(uint256 value) internal pure returns (uint72) {
   require(value <= type(uint72).max, "SafeCast: value doesn't fit in 72 bits");
   return uint72(value);
}

/**
 * @dev Returns the downcasted uint64 from uint256, reverting on
 * overflow (when the input is greater than largest uint64).
 *
 * Counterpart to Solidity's `uint64` operator.
 *
 * Requirements:
 *
 * - input must fit into 64 bits
 *
 * _Available since v2.5._
 */
function toUint64(uint256 value) internal pure returns (uint64) {
   require(value <= type(uint64).max, "SafeCast: value doesn't fit in 64 bits");
```

```
    return uint64(value);
}

/**
 * @dev Returns the downcasted uint56 from uint256, reverting on
 * overflow (when the input is greater than largest uint56).
 *
 * Counterpart to Solidity's `uint56` operator.
 *
 * Requirements:
 *
 * - input must fit into 56 bits
 *
 * _Available since v4.7._
 */
function toUint56(uint256 value) internal pure returns (uint56) {
    require(value <= type(uint56).max, "SafeCast: value doesn't fit in 56 bits");
    return uint56(value);
}

/**
 * @dev Returns the downcasted uint48 from uint256, reverting on
 * overflow (when the input is greater than largest uint48).
 *
 * Counterpart to Solidity's `uint48` operator.
 *
 * Requirements:
 *
 * - input must fit into 48 bits
 *
 * _Available since v4.7._
 */
function toUint48(uint256 value) internal pure returns (uint48) {
    require(value <= type(uint48).max, "SafeCast: value doesn't fit in 48 bits");
    return uint48(value);
}

/**
 * @dev Returns the downcasted uint40 from uint256, reverting on
 * overflow (when the input is greater than largest uint40).
 *
 * Counterpart to Solidity's `uint40` operator.
 *
 * Requirements:
 *
 * - input must fit into 40 bits
 *
 * _Available since v4.7._
 */
function toUint40(uint256 value) internal pure returns (uint40) {
    require(value <= type(uint40).max, "SafeCast: value doesn't fit in 40 bits");
    return uint40(value);
}

/**
 * @dev Returns the downcasted uint32 from uint256, reverting on
 * overflow (when the input is greater than largest uint32).
 *
 * Counterpart to Solidity's `uint32` operator.
 *
 * Requirements:
 *
 * - input must fit into 32 bits
 *
 * _Available since v2.5._
 */
function toUint32(uint256 value) internal pure returns (uint32) {
    require(value <= type(uint32).max, "SafeCast: value doesn't fit in 32 bits");
    return uint32(value);
}

/**
 * @dev Returns the downcasted uint24 from uint256, reverting on
 * overflow (when the input is greater than largest uint24).
 *
 * Counterpart to Solidity's `uint24` operator.
 *
 * Requirements:
 *
 * - input must fit into 24 bits
 *
 * _Available since v4.7._
 */
function toUint24(uint256 value) internal pure returns (uint24) {
    require(value <= type(uint24).max, "SafeCast: value doesn't fit in 24 bits");
    return uint24(value);
}

/**
 * @dev Returns the downcasted uint16 from uint256, reverting on
 * overflow (when the input is greater than largest uint16).
 *
 * Counterpart to Solidity's `uint16` operator.
 *
 * Requirements:
 *
 * - input must fit into 16 bits
 *
 * _Available since v2.5._
 */
function toUint16(uint256 value) internal pure returns (uint16) {
    require(value <= type(uint16).max, "SafeCast: value doesn't fit in 16 bits");
    return uint16(value);
}

/**
 * @dev Returns the downcasted uint8 from uint256, reverting on
 * overflow (when the input is greater than largest uint8).
 *
 * Counterpart to Solidity's `uint8` operator.
 *
 * Requirements:
 *
 * - input must fit into 8 bits
 *
 * _Available since v2.5._
 */
function toUint8(uint256 value) internal pure returns (uint8) {
    require(value <= type(uint8).max, "SafeCast: value doesn't fit in 8 bits");
    return uint8(value);
}

/**
 * @dev Converts a signed int256 into an unsigned uint256.
```

```
     *
     * Requirements:
     *
     * - input must be greater than or equal to 0.
     *
     * _Available since v3.0._
     */
    function toUint256(int256 value) internal pure returns (uint256) {
        require(value >= 0, "SafeCast: value must be positive");
        return uint256(value);
    }

    /**
     * @dev Returns the downcasted int248 from int256, reverting on
     * overflow (when the input is less than smallest int248 or
     * greater than largest int248).
     *
     * Counterpart to Solidity's `int248` operator.
     *
     * Requirements:
     *
     * - input must fit into 248 bits
     *
     * _Available since v4.7._
     */
    function toInt248(int256 value) internal pure returns (int248) {
        require(value >= type(int248).min && value <= type(int248).max, "SafeCast: value doesn't fit in 248 bits");
        return int248(value);
    }
    /**
     * @dev Returns the downcasted int240 from int256, reverting on
     * overflow (when the input is less than smallest int240 or
     * greater than largest int240).
     *
     * Counterpart to Solidity's `int240` operator.
     *
     * Requirements:
     *
     * - input must fit into 240 bits
     *
     * _Available since v4.7._
     */
    function toInt240(int256 value) internal pure returns (int240) {
        require(value >= type(int240).min && value <= type(int240).max, "SafeCast: value doesn't fit in 240 bits");
        return int240(value);
    }

    /**
     * @dev Returns the downcasted int232 from int256, reverting on
     * overflow (when the input is less than smallest int232 or
     * greater than largest int232).
     *
     * Counterpart to Solidity's `int232` operator.
     *
     * Requirements:
     *
     * - input must fit into 232 bits
     *
     * _Available since v4.7._
     */
    function toInt232(int256 value) internal pure returns (int232) {
        require(value >= type(int232).min && value <= type(int232).max, "SafeCast: value doesn't fit in
232 bits");
        return int232(value);
    }

    /**
     * @dev Returns the downcasted int224 from int256, reverting on
     * overflow (when the input is less than smallest int224 or
     * greater than largest int224).
     *
     * Counterpart to Solidity's `int224` operator.
     *
     * Requirements:
     *
     * - input must fit into 224 bits
     *
     * _Available since v4.7._
     */
    function toInt224(int256 value) internal pure returns (int224) {
        require(value >= type(int224).min && value <= type(int224).max, "SafeCast: value doesn't fit in
224 bits");
        return int224(value);
    }

    /**
     * @dev Returns the downcasted int216 from int256, reverting on
     * overflow (when the input is less than smallest int216 or
     * greater than largest int216).
     *
     * Counterpart to Solidity's `int216` operator.
     *
     * Requirements:
     *
     * - input must fit into 216 bits
     *
     * _Available since v4.7._
     */
    function toInt216(int256 value) internal pure returns (int216) {
        require(value >= type(int216).min && value <= type(int216).max, "SafeCast: value doesn't fit in 216 bits");
        return int216(value);
    }

    /**
     * @dev Returns the downcasted int208 from int256, reverting on
     * overflow (when the input is less than smallest int208 or
     * greater than largest int208).
     *
     * Counterpart to Solidity's `int208` operator.
     *
     * Requirements:
     *
     * - input must fit into 208 bits
     *
     * _Available since v4.7._
     */
    function toInt208(int256 value) internal pure returns (int208) {
        require(value >= type(int208).min && value <= type(int208).max, "SafeCast: value doesn't fit in 208 bits");
        return int208(value);
    }
    /**
     * @dev Returns the downcasted int200 from int256, reverting on
     * overflow (when the input is less than smallest int200 or
     * greater than largest int200).
     *
     * Counterpart to Solidity's `int200` operator.
     *
     * Requirements:
     *
     * - input must fit into 200 bits
     *
     * _Available since v4.7._
     */
    function toInt200(int256 value) internal pure returns (int200) {
        require(value >= type(int200).min && value <= type(int200).max, "SafeCast: value doesn't fit in 200 bits");
        return int200(value);
    }

    /**
     * @dev Returns the downcasted int192 from int256, reverting on
     * overflow (when the input is less than smallest int192 or
     * greater than largest int192).
     *
     * Counterpart to Solidity's `int192` operator.
     *
     * Requirements:
     *
     * - input must fit into 192 bits
     *
     * _Available since v4.7._
     */
    function toInt192(int256 value) internal pure returns (int192) {
        require(value >= type(int192).min && value <= type(int192).max, "SafeCast: value doesn't fit in 192 bits");
        return int192(value);
    }

    /**
     * @dev Returns the downcasted int184 from int256, reverting on
     * overflow (when the input is less than smallest int184 or
     * greater than largest int184).
     *
     * Counterpart to Solidity's `int184` operator.
     *
     * Requirements:
     *
     * - input must fit into 184 bits
     *
     * _Available since v4.7._
     */
    function toInt184(int256 value) internal pure returns (int184) {
        require(value >= type(int184).min && value <= type(int184).max, "SafeCast: value doesn't fit in 184 bits");
        return int184(value);
    }

    /**
     * @dev Returns the downcasted int176 from int256, reverting on
     * overflow (when the input is less than smallest int176 or
     * greater than largest int176).
     *
     * Counterpart to Solidity's `int176` operator.
     *
     * Requirements:
     *
     * - input must fit into 176 bits
     *
     * _Available since v4.7._
     */
    function toInt176(int256 value) internal pure returns (int176) {
        require(value >= type(int176).min && value <= type(int176).max, "SafeCast: value doesn't fit in 176 bits");
        return int176(value);
    }

    /**
     * @dev Returns the downcasted int168 from int256, reverting on
     * overflow (when the input is less than smallest int168 or
     * greater than largest int168).
     *
     * Counterpart to Solidity's `int168` operator.
     *
     * Requirements:
     *
     * - input must fit into 168 bits
     *
     * _Available since v4.7._
     */
    function toInt168(int256 value) internal pure returns (int168) {
        require(value >= type(int168).min && value <= type(int168).max, "SafeCast: value doesn't fit in 168 bits");
        return int168(value);
    }
    /**
     * @dev Returns the downcasted int160 from int256, reverting on
     * overflow (when the input is less than smallest int160 or
     * greater than largest int160).
     *
     * Counterpart to Solidity's `int160` operator.
     *
     * Requirements:
     *
     * - input must fit into 160 bits
     *
     * _Available since v4.7._
     */
    function toInt160(int256 value) internal pure returns (int160) {
        require(value >= type(int160).min && value <= type(int160).max, "SafeCast: value doesn't fit in 160 bits");
        return int160(value);
    }

    /**
     * @dev Returns the downcasted int152 from int256, reverting on
     * overflow (when the input is less than smallest int152 or
     * greater than largest int152).
     *
     * Counterpart to Solidity's `int152` operator.
     *
     * Requirements:
     *
     * - input must fit into 152 bits
     *
     * _Available since v4.7._
     */
    function toInt152(int256 value) internal pure returns (int152) {
        require(value >= type(int152).min && value <= type(int152).max, "SafeCast: value doesn't fit in 152 bits");
        return int152(value);
    }

    /**
     * @dev Returns the downcasted int144 from int256, reverting on
     * overflow (when the input is less than smallest int144 or
     * greater than largest int144).
     *
     * Counterpart to Solidity's `int144` operator.
     *
     * Requirements:
     *
     * - input must fit into 144 bits
     *
     * _Available since v4.7._
     */
    function toInt144(int256 value) internal pure returns (int144) {
        require(value >= type(int144).min && value <= type(int144).max, "SafeCast: value doesn't fit in 144 bits");
        return int144(value);
    }

    /**
     * @dev Returns the downcasted int136 from int256, reverting on
     * overflow (when the input is less than smallest int136 or
     * greater than largest int136).
     *
     * Counterpart to Solidity's `int136` operator.
     *
     * Requirements:
     *
     * - input must fit into 136 bits
     *
     * _Available since v4.7._
     */
    function toInt136(int256 value) internal pure returns (int136) {
        require(value >= type(int136).min && value <= type(int136).max, "SafeCast: value doesn't fit in 136 bits");
        return int136(value);
    }

    /**
     * @dev Returns the downcasted int128 from int256, reverting on
     * overflow (when the input is less than smallest int128 or
     * greater than largest int128).
     *
     * Counterpart to Solidity's `int128` operator.
     *
     * Requirements:
     *
     * - input must fit into 128 bits
     *
     * _Available since v3.1._
     */
    function toInt128(int256 value) internal pure returns (int128) {
        require(value >= type(int128).min && value <= type(int128).max, "SafeCast: value doesn't fit in 128 bits");
        return int128(value);
    }
    /**
     * @dev Returns the downcasted int120 from int256, reverting on
     * overflow (when the input is less than smallest int120 or
     * greater than largest int120).
     *
     * Counterpart to Solidity's `int120` operator.
     *
     * Requirements:
     *
     * - input must fit into 120 bits
     *
     * _Available since v4.7._
     */
    function toInt120(int256 value) internal pure returns (int120) {
        require(value >= type(int120).min && value <= type(int120).max, "SafeCast: value doesn't fit in 120 bits");
        return int120(value);
    }

    /**
     * @dev Returns the downcasted int112 from int256, reverting on
     * overflow (when the input is less than smallest int112 or
     * greater than largest int112).
     *
     * Counterpart to Solidity's `int112` operator.
     *
     * Requirements:
     *
     * - input must fit into 112 bits
     *
     * _Available since v4.7._
     */
    function toInt112(int256 value) internal pure returns (int112) {
        require(value >= type(int112).min && value <= type(int112).max, "SafeCast: value doesn't fit in 112 bits");
        return int112(value);
    }

    /**
     * @dev Returns the downcasted int104 from int256, reverting on
     * overflow (when the input is less than smallest int104 or
     * greater than largest int104).
     *
     * Counterpart to Solidity's `int104` operator.
     *
     * Requirements:
     *
     * - input must fit into 104 bits
     *
     * _Available since v4.7._
     */
    function toInt104(int256 value) internal pure returns (int104) {
        require(value >= type(int104).min && value <= type(int104).max, "SafeCast: value doesn't fit in 104 bits");
        return int104(value);
    }

    /**
     * @dev Returns the downcasted int96 from int256, reverting on
     * overflow (when the input is less than smallest int96 or
     * greater than largest int96).
     *
     * Counterpart to Solidity's `int96` operator.
     *
     * Requirements:
     *
     * - input must fit into 96 bits
     *
     * _Available since v4.7._
     */
    function toInt96(int256 value) internal pure returns (int96) {
        require(value >= type(int96).min && value <= type(int96).max, "SafeCast: value doesn't fit in 96
bits");
        return int96(value);
    }

    /**
     * @dev Returns the downcasted int88 from int256, reverting on
     * overflow (when the input is less than smallest int88 or
     * greater than largest int88).
     *
     * Counterpart to Solidity's `int88` operator.
     *
     * Requirements:
     *
     * - input must fit into 88 bits
     *
     * _Available since v4.7._
     */
    function toInt88(int256 value) internal pure returns (int88) {
        require(value >= type(int88).min && value <= type(int88).max, "SafeCast: value doesn't fit in 88
bits");
        return int88(value);
    }
    /**
     * @dev Returns the downcasted int80 from int256, reverting on
     * overflow (when the input is less than smallest int80 or
     * greater than largest int80).
     *
     * Counterpart to Solidity's `int80` operator.
     *
     * Requirements:
     *
     * - input must fit into 80 bits
     *
     * _Available since v4.7._
     */
    function toInt80(int256 value) internal pure returns (int80) {
        require(value >= type(int80).min && value <= type(int80).max, "SafeCast: value doesn't fit in 80 bits");
        return int80(value);
    }

    /**
     * @dev Returns the downcasted int72 from int256, reverting on
     * overflow (when the input is less than smallest int72 or
     * greater than largest int72).
     *
     * Counterpart to Solidity's `int72` operator.
     *
     * Requirements:
     *
     * - input must fit into 72 bits
     *
     * _Available since v4.7._
     */
    function toInt72(int256 value) internal pure returns (int72) {
        require(value >= type(int72).min && value <= type(int72).max, "SafeCast: value doesn't fit in 72 bits");
        return int72(value);
    }

    /**
     * @dev Returns the downcasted int64 from int256, reverting on
     * overflow (when the input is less than smallest int64 or
     * greater than largest int64).
     *
     * Counterpart to Solidity's `int64` operator.
     *
     * Requirements:
     *
     * - input must fit into 64 bits
     *
     * _Available since v3.1._
     */
    function toInt64(int256 value) internal pure returns (int64) {
        require(value >= type(int64).min && value <= type(int64).max, "SafeCast: value doesn't fit in 64 bits");
        return int64(value);
    }

    /**
     * @dev Returns the downcasted int56 from int256, reverting on
     * overflow (when the input is less than smallest int56 or
     * greater than largest int56).
     *
     * Counterpart to Solidity's `int56` operator.
     *
     * Requirements:
     *
     * - input must fit into 56 bits
     *
     * _Available since v4.7._
     */
    function toInt56(int256 value) internal pure returns (int56) {
        require(value >= type(int56).min && value <= type(int56).max, "SafeCast: value doesn't fit in 56 bits");
        return int56(value);
    }

    /**
     * @dev Returns the downcasted int48 from int256, reverting on
     * overflow (when the input is less than smallest int48 or
     * greater than largest int48).
     *
     * Counterpart to Solidity's `int48` operator.
     *
     * Requirements:
     *
     * - input must fit into 48 bits
     *
     * _Available since v4.7._
     */
    function toInt48(int256 value) internal pure returns (int48) {
        require(value >= type(int48).min && value <= type(int48).max, "SafeCast: value doesn't fit in 48 bits");
        return int48(value);
    }
    /**
     * @dev Returns the downcasted int40 from int256, reverting on
     * overflow (when the input is less than smallest int40 or
     * greater than largest int40).
     *
     * Counterpart to Solidity's `int40` operator.
     *
     * Requirements:
     *
     * - input must fit into 40 bits
     *
     * _Available since v4.7._
     */
    function toInt40(int256 value) internal pure returns (int40) {
        require(value >= type(int40).min && value <= type(int40).max, "SafeCast: value doesn't fit in 40 bits");
        return int40(value);
    }

    /**
     * @dev Returns the downcasted int32 from int256, reverting on
     * overflow (when the input is less than smallest int32 or
     * greater than largest int32).
     *
     * Counterpart to Solidity's `int32` operator.
     *
     * Requirements:
     *
     * - input must fit into 32 bits
     *
     * _Available since v3.1._
     */
    function toInt32(int256 value) internal pure returns (int32) {
        require(value >= type(int32).min && value <= type(int32).max, "SafeCast: value doesn't fit in 32
bits");
        return int32(value);
    }

    /**
     * @dev Returns the downcasted int24 from int256, reverting on
     * overflow (when the input is less than smallest int24 or
     * greater than largest int24).
     *
     * Counterpart to Solidity's `int24` operator.
     *
     * Requirements:
     *
     * - input must fit into 24 bits
     *
     * _Available since v4.7._
     */
    function toInt24(int256 value) internal pure returns (int24) {
        require(value >= type(int24).min && value <= type(int24).max, "SafeCast: value doesn't fit in 24
bits");
        return int24(value);
    }

    /**
     * @dev Returns the downcasted int16 from int256, reverting on
     * overflow (when the input is less than smallest int16 or
     * greater than largest int16).
     *
     * Counterpart to Solidity's `int16` operator.
     *
     * Requirements:
     *
     * - input must fit into 16 bits
     *
     * _Available since v3.1._
     */
    function toInt16(int256 value) internal pure returns (int16) {
        require(value >= type(int16).min && value <= type(int16).max, "SafeCast: value doesn't fit in 16 bits");
        return int16(value);
    }

    /**
     * @dev Returns the downcasted int8 from int256, reverting on
     * overflow (when the input is less than smallest int8 or
     * greater than largest int8).
     *
     * Counterpart to Solidity's `int8` operator.
     *
     * Requirements:
     *
     * - input must fit into 8 bits
     *
     * _Available since v3.1._
     */
    function toInt8(int256 value) internal pure returns (int8) {
        require(value >= type(int8).min && value <= type(int8).max, "SafeCast: value doesn't fit in 8 bits");
        return int8(value);
    }
    /**
     * @dev Converts an unsigned uint256 into a signed int256.
     *
     * Requirements:
     *
     * - input must be less than or equal to maxInt256.
     *
     * _Available since v3.0._
     */
    function toInt256(uint256 value) internal pure returns (int256) {
        // Note: Unsafe cast below is okay because `type(int256).max` is guaranteed to be positive
        require(value <= uint256(type(int256).max), "SafeCast: value doesn't fit in an int256");
        return int256(value);
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.5.0) (governance/utils/IVotes.sol)
pragma solidity ^0.8.0;

/**
 * @dev Common interface for {ERC20Votes}, {ERC721Votes}, and other {Votes}-enabled contracts.
 *
 * _Available since v4.5._
 */
interface IVotes {
    /**
     * @dev Emitted when an account changes their delegate.
     */
    event DelegateChanged(address indexed delegator, address indexed fromDelegate, address
indexed toDelegate);
```

```
    /**
     * @dev Emitted when a token transfer or delegate change results in changes to a delegate's number of votes.
     */
    event DelegateVotesChanged(address indexed delegate, uint256 previousBalance, uint256 newBalance);

    /**
     * @dev Returns the current amount of votes that `account` has.
     */
    function getVotes(address account) external view returns (uint256);

    /**
     * @dev Returns the amount of votes that `account` had at the end of a past block (`blockNumber`).
     */
    function getPastVotes(address account, uint256 blockNumber) external view returns (uint256);

    /**
     * @dev Returns the total supply of votes available at the end of a past block (`blockNumber`).
     *
     * NOTE: This value is the sum of all available votes, which is not necessarily the sum of all delegated votes.
     * Votes that have not been delegated are still part of total supply, even though they would not participate in a
     * vote.
     */
    function getPastTotalSupply(uint256 blockNumber) external view returns (uint256);

    /**
     * @dev Returns the delegate that `account` has chosen.
     */
    function delegates(address account) external view returns (address);
    /**
     * @dev Delegates votes from the sender to `delegatee`.
     */
    function delegate(address delegatee) external;

    /**
     * @dev Delegates votes from signer to `delegatee`.
     */
    function delegateBySig(
        address delegatee,
        uint256 nonce,
        uint256 expiry,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0) (utils/math/Math.sol)

pragma solidity ^0.8.0;

/**
 * @dev Standard math utilities missing in the Solidity language.
 */
library Math {
    enum Rounding {
        Down, // Toward negative infinity
        Up, // Toward infinity
        Zero // Toward zero
    }

    /**
     * @dev Returns the largest of two numbers.
     */
    function max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a >= b ? a : b;
    }

    /**
     * @dev Returns the smallest of two numbers.
     */
    function min(uint256 a, uint256 b) internal pure returns (uint256) {
        return a < b ? a : b;
    }

    /**
     * @dev Returns the average of two numbers. The result is rounded towards
     * zero.
     */
    function average(uint256 a, uint256 b) internal pure returns (uint256) {
        // (a + b) / 2 can overflow.
        return (a & b) + (a ^ b) / 2;
    }

    /**
     * @dev Returns the ceiling of the division of two numbers.
     *
     * This differs from standard division with `/` in that it rounds up instead
     * of rounding down.
     */
```

```
    function ceilDiv(uint256 a, uint256 b) internal pure returns (uint256) {
        // (a + b - 1) / b can overflow on addition, so we distribute.
        return a == 0 ? 0 : (a - 1) / b + 1;
    }

    /**
     * @notice Calculates floor(x * y / denominator) with full precision. Throws if result overflows a uint256 or denominator == 0
     * @dev Original credit to Remco Bloemen under MIT license (https://xn--2-umb.com/21/muldiv)
     * with further edits by Uniswap Labs also under MIT license.
     */
    function mulDiv(
        uint256 x,
        uint256 y,
        uint256 denominator
    ) internal pure returns (uint256 result) {
        unchecked {
            // 512-bit multiply [prod1 prod0] = x * y. Compute the product mod 2^256 and mod 2^256 - 1, then use
            // use the Chinese Remainder Theorem to reconstruct the 512 bit result. The result is stored in two 256
            // variables such that product = prod1 * 2^256 + prod0.
            uint256 prod0; // Least significant 256 bits of the product
            uint256 prod1; // Most significant 256 bits of the product
            assembly {
                let mm := mulmod(x, y, not(0))
                prod0 := mul(x, y)
                prod1 := sub(sub(mm, prod0), lt(mm, prod0))
            }

            // Handle non-overflow cases, 256 by 256 division.
            if (prod1 == 0) {
```

```
    return prod0 / denominator;
}

// Make sure the result is less than 2^256. Also prevents denominator == 0.
require(denominator > prod1);

///////////////////////////////////////////////
// 512 by 256 division.
///////////////////////////////////////////////

// Make division exact by subtracting the remainder from [prod1 prod0].
uint256 remainder;
assembly {
    // Compute remainder using mulmod.
    remainder := mulmod(x, y, denominator)

    // Subtract 256 bit number from 512 bit number.
    prod1 := sub(prod1, gt(remainder, prod0))
    prod0 := sub(prod0, remainder)
}

// Factor powers of two out of denominator and compute largest power of two divisor of denominator. Always >= 1.
// See https://cs.stackexchange.com/q/138556/92363.

// Does not overflow because the denominator cannot be zero at this stage in the function.
uint256 twos = denominator & (~denominator + 1);
assembly {
    // Divide denominator by twos.
    denominator := div(denominator, twos)
```

```
        // Divide [prod1 prod0] by twos.
        prod0 := div(prod0, twos)

        // Flip twos such that it is 2^256 / twos. If twos is zero, then it becomes one.
        twos := add(div(sub(0, twos), twos), 1)
    }

    // Shift in bits from prod1 into prod0.
    prod0 |= prod1 * twos;

    // Invert denominator mod 2^256. Now that denominator is an odd number, it has an inverse modulo 2^256 such
    // that denominator * inv = 1 mod 2^256. Compute the inverse by starting with a seed that is correct for
    // four bits. That is, denominator * inv = 1 mod 2^4.
    uint256 inverse = (3 * denominator) ^ 2;

    // Use the Newton-Raphson iteration to improve the precision. Thanks to Hensel's lifting lemma, this also works
    // in modular arithmetic, doubling the correct bits in each step.
    inverse *= 2 - denominator * inverse; // inverse mod 2^8
    inverse *= 2 - denominator * inverse; // inverse mod 2^16
    inverse *= 2 - denominator * inverse; // inverse mod 2^32
    inverse *= 2 - denominator * inverse; // inverse mod 2^64
    inverse *= 2 - denominator * inverse; // inverse mod 2^128
    inverse *= 2 - denominator * inverse; // inverse mod 2^256

    // Because the division is now exact we can divide by multiplying with the modular inverse of denominator.
    // This will give us the correct result modulo 2^256. Since the preconditions guarantee that the outcome is
    // less than 2^256, this is the final result. We don't need to compute the high bits of the result and prod1
```

```
            // is no longer required.
            result = prod0 * inverse;
            return result;
        }
    }

    /**
     * @notice Calculates x * y / denominator with full precision, following the selected rounding direction.
     */
    function mulDiv(
        uint256 x,
        uint256 y,
        uint256 denominator,
        Rounding rounding
    ) internal pure returns (uint256) {
        uint256 result = mulDiv(x, y, denominator);
        if (rounding == Rounding.Up && mulmod(x, y, denominator) > 0) {
            result += 1;
        }
        return result;
    }

    /**
     * @dev Returns the square root of a number. It the number is not a perfect square, the value is rounded down.
     *
     * Inspired by Henry S. Warren, Jr.'s "Hacker's Delight" (Chapter 11).
     */
    function sqrt(uint256 a) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        // For our first guess, we get the biggest power of 2 which is smaller than the square root of the
    target.
        // We know that the "msb" (most significant bit) of our target number `a` is a power of 2 such
    that we have
        // `msb(a) <= a < 2*msb(a)`.
        // We also know that `k`, the position of the most significant bit, is such that `msb(a) = 2**k`.
        // This gives `2**k < a <= 2**(k+1)` → `2**(k/2) <= sqrt(a) < 2 ** (k/2+1)`.
        // Using an algorithm similar to the msb conmputation, we are able to compute `result =
    2**(k/2)` which is a
        // good first aproximation of `sqrt(a)` with at least 1 correct bit.
        uint256 result = 1;
        uint256 x = a;
        if (x >> 128 > 0) {
            x >>= 128;
            result <<= 64;
        }
        if (x >> 64 > 0) {
            x >>= 64;
            result <<= 32;
        }
        if (x >> 32 > 0) {
            x >>= 32;
            result <<= 16;
        }
        if (x >> 16 > 0) {
            x >>= 16;
            result <<= 8;
        }
        if (x >> 8 > 0) {
            x >>= 8;
```

```
            result <<= 4;
        }
        if (x >> 4 > 0) {
            x >>= 4;
            result <<= 2;
        }
        if (x >> 2 > 0) {
            result <<= 1;
        }

        // At this point `result` is an estimation with one bit of precision. We know the true value is a
uint128,
        // since it is the square root of a uint256. Newton's method converges quadratically (precision
doubles at
        // every iteration). We thus need at most 7 iteration to turn our partial result with one bit of
precision
        // into the expected uint128 result.
        unchecked {
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            return min(result, a / result);
        }
    }

    /**
     * @notice Calculates sqrt(a), following the selected rounding direction.
     */
    function sqrt(uint256 a, Rounding rounding) internal pure returns (uint256) {
        uint256 result = sqrt(a);
        if (rounding == Rounding.Up && result * result < a) {
            result += 1;
        }
        return result;
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (utils/Arrays.sol)

pragma solidity ^0.8.0;

import "./math/Math.sol";

/**
 * @dev Collection of functions related to array types.
 */
library Arrays {
    /**
     * @dev Searches a sorted `array` and returns the first index that contains
     * a value greater or equal to `element`. If no such index exists (i.e. all
     * values in the array are strictly less than `element`), the array length is
     * returned. Time complexity O(log n).
     *
     * `array` is expected to be sorted in ascending order, and to contain no
     * repeated elements.
     */
    function findUpperBound(uint256[] storage array, uint256 element) internal view returns (uint256)
{
        if (array.length == 0) {
            return 0;
        }

        uint256 low = 0;
        uint256 high = array.length;

        while (low < high) {
            uint256 mid = Math.average(low, high);

            // Note that mid will always be strictly less than high (i.e. it will be a valid array index)
            // because Math.average rounds down (it does integer division with truncation).
            if (array[mid] > element) {
                high = mid;
            } else {
                low = mid + 1;
            }
        }

        // At this point `low` is the exclusive upper bound. We will return the inclusive upper bound.
        if (low > 0 && array[low - 1] == element) {
            return low - 1;
        } else {
            return low;
        }
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (utils/Context.sol)

pragma solidity ^0.8.0;
```

```
/**
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes calldata) {
        return msg.data;
    }
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (token/ERC20/extensions/IERC20Metadata.sol)

pragma solidity ^0.8.0;

import "../IERC20.sol";

/**
 * @dev Interface for the optional metadata functions from the ERC20 standard.
 *
 * _Available since v4.1._
 */
```

```
interface IERC20Metadata is IERC20 {
    /**
     * @dev Returns the name of the token.
     */
    function name() external view returns (string memory);

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

    /**
     * @dev Returns the decimals places of the token.
     */
    function decimals() external view returns (uint8);
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.6.0) (token/ERC20/IERC20.sol)

pragma solidity ^0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
```

```
event Transfer(address indexed from, address indexed to, uint256 value);

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

/**
 * @dev Returns the amount of tokens in existence.
 */
function totalSupply() external view returns (uint256);

/**
 * @dev Returns the amount of tokens owned by `account`.
 */
function balanceOf(address account) external view returns (uint256);

/**
 * @dev Moves `amount` tokens from the caller's account to `to`.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * Emits a {Transfer} event.
 */
function transfer(address to, uint256 amount) external returns (bool);

/**
 * @dev Returns the remaining number of tokens that `spender` will be
 * allowed to spend on behalf of `owner` through {transferFrom}. This is
 * zero by default.
 *
 * This value changes when {approve} or {transferFrom} are called.
 */
function allowance(address owner, address spender) external view returns (uint256);

/**
 * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * IMPORTANT: Beware that changing an allowance with this method brings the risk
 * that someone may use both the old and the new allowance by unfortunate
 * transaction ordering. One possible solution to mitigate this race
 * condition is to first reduce the spender's allowance to 0 and set the
 * desired value afterwards:
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
 *
 * Emits an {Approval} event.
 */
function approve(address spender, uint256 amount) external returns (bool);

/**
 * @dev Moves `amount` tokens from `from` to `to` using the
 * allowance mechanism. `amount` is then deducted from the caller's
 * allowance.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * Emits a {Transfer} event.
 */
function transferFrom(
        address from,
        address to,
        uint256 amount
    ) external returns (bool);
}
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0) (utils/Strings.sol)

pragma solidity ^0.8.0;

/**
 * @dev String operations.
 */
library Strings {
    bytes16 private constant _HEX_SYMBOLS = "0123456789abcdef";
    uint8 private constant _ADDRESS_LENGTH = 20;

    /**
     * @dev Converts a `uint256` to its ASCII `string` decimal representation.
     */
    function toString(uint256 value) internal pure returns (string memory) {
        // Inspired by OraclizeAPI's implementation - MIT licence
        // https://github.com/oraclize/ethereum-api/blob/b42146b063c7d6ee1358846c198246239e9360e8/oraclizeAPI_0.4.25.sol

        if (value == 0) {
            return "0";
        }
        uint256 temp = value;
        uint256 digits;
        while (temp != 0) {
```

```
        digits++;
        temp /= 10;
    }
    bytes memory buffer = new bytes(digits);
    while (value != 0) {
        digits -= 1;
        buffer[digits] = bytes1(uint8(48 + uint256(value % 10)));
        value /= 10;
    }
    return string(buffer);
}

/**
 * @dev Converts a `uint256` to its ASCII `string` hexadecimal representation.
 */
function toHexString(uint256 value) internal pure returns (string memory) {
    if (value == 0) {
        return "0x00";
    }
    uint256 temp = value;
    uint256 length = 0;
    while (temp != 0) {
        length++;
        temp >>= 8;
    }
    return toHexString(value, length);
}

/**
 * @dev Converts a `uint256` to its ASCII `string` hexadecimal representation with fixed length.
 */
```

```
    function toHexString(uint256 value, uint256 length) internal pure returns (string memory) {
        bytes memory buffer = new bytes(2 * length + 2);
        buffer[0] = "0";
        buffer[1] = "x";
        for (uint256 i = 2 * length + 1; i > 1; --i) {
            buffer[i] = _HEX_SYMBOLS[value & 0xf];
            value >>= 4;
        }
        require(value == 0, "Strings: hex length insufficient");
        return string(buffer);
    }

    /**
     * @dev Converts an `address` with fixed length of 20 bytes to its not checksummed ASCII `string` hexadecimal representation.
     */
    function toHexString(address addr) internal pure returns (string memory) {
        return toHexString(uint256(uint160(addr)), _ADDRESS_LENGTH);
    }
}
```

```
{
  "optimizer": {
    "enabled": false,
    "runs": 200
  },
  "outputSelection": {
    "*": {
      "*": [
        "evm.bytecode",
        "evm.deployedBytecode",
        "devdoc",
        "userdoc",
        "metadata",
        "abi"
      ]
    }
  }
}
```

Contract ABI

[{"inputs":[],"stateMutability":"nonpayable","type":"constructor"},{"anonymous":false,"inputs":[{"indexed":true,"internalType":"address","name":"owner","type":"address"},{"indexed":true,"internalType":"address","name":"spender","type":"address"},{"indexed":false,"internalType":"uint256","name":"value","type":"uint256"}],"name":"Approval","type":"event"},{"anonymous":false,"inputs":[{"indexed":true,"internalType":"address","name":"delegator","type":"address"},{"indexed":true,"internalType":"address","name":"fromDelegate","type":"address"},{"indexed":true,"internalType":"address","name":"toDelegate","type":"address"}],"name":"DelegateChanged","type":"event"},{"anonymous":false,"inputs":[{"indexed":true,"internalType":"address","name":"delegate","type":"address"},{"indexed":false,"internalType":"uint256","name":"previousBalance","type":"uint256"},{"indexed":false,"internalType":"uint256","name":"newBalance","type":"uint256"}],"name":"DelegateVotesChanged","type":"event"},{"anonymous":false,"inputs":[{"indexed":true,"internalType":"address","name":"previousOwner","type":"address"},{"indexed":true,"internalType":"address","name":"newOwner","type":"address"}],"name":"OwnershipTransferred","type":"event"},{"anonymous":false,"inputs":[{"indexed":false,"internalType":"uint256","name":"id","type":"uint256"}],"name":"Snapshot","type":"event"},{"anonymous":false,"inputs":[{"indexed":true,"internalType":"address","name":"from","type":"address"},{"indexed":true,"internalType":"address","name":"to","type":"address"},{"indexed":false,"internalType":"uint256","name":"value","type":"uint256"}],"name":"Transfer","type":"event"},{"inputs":[],"name":"DOMAIN_SEPARATOR","outputs":[{"internalType":"bytes32","name":"","type":"bytes32"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"owner","type":"address"},{"internalType":"address","name":"spender","type":"address"}],"name":"allowance","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"spender","type":"address"},{"internalType":"uint256","name":"amount","type":"uint256"}],"name":"approve","outputs":[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"}],"name":"balanceOf","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"},{"internalType":"uint256","name":"snapshotId","type":"uint256"}],"name":"balanceOfAt","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"uint256","name":"amount","type":"uint256"}],"name":"burn","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"},{"internalType":"uint256","name":"amount","type":"uint256"}],"name":"burnFrom","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"},{"internalType":"uint32","name":"pos","type":"uint32"}],"name":"checkpoints","outputs":[{"components":[{"internalType":"uint32","name":"fromBlock","type":"uint32"},{"internalType":"uint224","name":"votes","type":"uint224"}],"internalType":"struct ERC20Votes.Checkpoint","name":"","type":"tuple"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"decimals","outputs":[{"internalType":"uint8","name":"","type":"uint8"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"spender","type":"address"},{"internalType":"uint256","name":"subtractedValue","type":"uint256"}],"name":"decreaseAllowance","outputs":[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"delegatee","type":"address"}],"name":"delegate","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"delegatee","type":"address"},{"internalType":"uint256","name":"nonce","type":"uint256"},{"internalType":"uint256","name":"expiry","type":"uint256"},{"internalType":"uint8","name":"v","type":"uint8"},{"internalType":"bytes

32","name":"r","type":"bytes32"},{"internalType":"bytes32","name":"s","type":"bytes32"}],"name":"delegateBySig","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"}],"name":"delegates","outputs":[{"internalType":"address","name":"","type":"address"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"uint256","name":"blockNumber","type":"uint256"}],"name":"getPastTotalSupply","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"},{"internalType":"uint256","name":"blockNumber","type":"uint256"}],"name":"getPastVotes","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"}],"name":"getVotes","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"spender","type":"address"},{"internalType":"uint256","name":"addedValue","type":"uint256"}],"name":"increaseAllowance","outputs":[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[],"name":"name","outputs":[{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"owner","type":"address"}],"name":"nonces","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"account","type":"address"}],"name":"numCheckpoints","outputs":[{"internalType":"uint32","name":"","type":"uint32"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"owner","outputs":[{"internalType":"address","name":"","type":"address"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"owner","type":"address"},{"internalType":"address","name":"spender","type":"address"},{"internalType":"uint256","name":"value","type":"uint256"},{"internalType":"uint256","name":"deadline","type":"uint256"},{"internalType":"uint8","name":"v","type":"uint8"},{"internalType":"bytes32","name":"r","type":"bytes32"},{"internalType":"bytes32","name":"s","type":"bytes32"}],"name":"permit","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[],"name":"renounceOwnership","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[],"name":"snapshot","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[],"name":"symbol","outputs":[{"internalType":"string","name":"","type":"string"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"bool","name":"enable","type":"bool"}],"name":"toggleBurn","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[],"name":"totalSupply","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"uint256","name":"snapshotId","type":"uint256"}],"name":"totalSupplyAt","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"amount","type":"uint256"}],"name":"transfer","outputs":[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"from","type":"address"},{"internalType":"address","name":"to","type":"address"},{"internalType":"uint256","name":"amount","type":"uint256"}],"name":"transferFrom","outputs":[{"internalType":"bool","name":"","type":"bool"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"newOwner","type":"address"}],"name":"transferOwnership","outputs":[],"stateMutability":"nonpayable","type":"function"}]

Contract Creation Code

```
6101406040526001600f60006101000a81548160ff0219169083151502179055503480156200002d576000
80fd5b5060405180604001604052806006815260200017f582d4c594e580000000000000000000000000000
0000000000000000000000008152508060405180604001604052806001815260200017f310000000000000
000000000000000000000000000000000000000000000000008152506040518060400160405280600681526 0
20017f582d4c594e580000000000000000000000000000000000000000000000000008152506040518060
40016040528060048152602001 7f4c594e5300000000000000000000000000000000000000000000000000
0000008152508160039081620001189190620011b8565b5080600490816200012a9190620011b8565b5050
506200014d620001416200024660201b60201c565b6200024e60201b60201c565b60008280519060200120
9050600082805190602001209050600 07f8b73c3c69bb8fe3d512ecc4cf759cc79239f7b179b0ffacaa9a7
5d522b39400f90508260e081815250508161010081815250504660a0818152505062000 1b6818484620003
1460201b60201c565b60808181525050307 3ffffffffffffffffffffffffffffffffffffffff1660c08173
ffffffffffffffffffffffffffffffffffffffff1681525050806101208181525050505050505050620002
4033620002156200035060201b60201c565b600a62000223919062000142f565b637d2b75006200023491 90
62001480565b6200035960201b60201c565b620018d3565b60003390509056 5b600060096000905490610 1
000a900473ffffffffffffffffffffffffffffffffffffffff16905081600960006101000a81548173ffff
ffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffff
```

```
ff1602179055508173ffffffffffffffffffffffffffffffffffffffff168173ffffffffffffffffffffff
ffffffffffffffffff167f8be0079c531659141344cd1fd0a4f28419497f9722a3daafe3b4186f6b6457e0
60405160405180910390a35050565b60008383834630604051602001620003319594939291906200153c56
5b6040516020818303038152906040528051906020012090509392505050565b60006012905090565b6200
036b82826200036f60201b60201c565b5050565b6200038182826200042860201b60201c565b6200039162
0005a060201b60201c565b7bffffffffffffffffffffffffffffffffffffffffffffffffffffffff166200
03bf620005c460201b60201c565b111562000403576040517f08c379a00000000000000000000000000000
00000000000000000000000000008152600401620003fa9062001620565b60405180910390fd5b62000422
600e620005ce60201b620011ee1783620005e660201b60201c565b5050505056 5b600073ffffffffffffff
ffffffffffffffffffffffffff168273ffffffffffffffffffffffffffffffffffffffff16036200049a57
6040517f08c379a00000000000000000000000000000000000000000000000000000000081526004016200
04919062001692565b60405180910390fd5b620004ae60008383620008886020 1b60201c565b8060026000
828254620004c29190620016b4565b92505081905550806000808473ffffffffffffffffffffffffffffff
ffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060
00828254620005199190620016b4565b9250508190555081 73ffffffffffffffffffffffffffffffffffff
ffff16600073ffffffffffffffffffffffffffffffffffffffff167fddf252ad1be2c89b69c2b068fc378d
aa952ba7f163c4a11628f55a4df523b3ef8360405162000580919062001 6ef565b60405180910390a36200
059c600083836200 08a060201b60201c565b5050565b60007bffffffffffffffffffffffffffffffffffff
ffffffffffffffffffff905090565b600060025490509056 5b60008183620005de9190620016b4565b9050
92915050565b600080600085805490509050600081146200065b57856001826200060b91906200170c565b
815481106200061f576200061e62001747565b5b9060005260206000200160000160049054906101000a90
047bffffffffffffffffffffffffffffffffffffffffffffffffffffffff166200065e565b60005b7bffff
ffffffffffffffffffffffffffffffffffffffffffffffffffff1692506200068a83858760201c565b9150
600081118015620006e35750438660018362 0006a991906200170c565b8154811062 0006bd57620006bc62
001747565b5b906000526020600020016000016000905490 6101000a900463ffffffff1663ffffffff1614
5b156200077f57620006fa82620008b860201b60201c565b866001836200070a91906200170c565b815481
106200071e576200071d62001747565b5b9060005260206000200160000160046101000a8154817bffffff
ffffffffffffffffffffffffffffffffffffffffffffffffff02191690837bffffffffffffffffffffffff
ffffffffffffffffffffffffffffffff1602179055506200087f565b85604051806040016040528062000 7
9c4362000926 60201b60201c565b63ffffffff168152602001620007b885620008b860201b60201c565b7b
ffffffffffffffffffffffffffffffffffffffffffffffffffffffff168152509080600181540180825580
915050600190039060005260206000200160009091909190915060008201518160000160006101000a8154
8163ffffffff021916908363ffffffff16021790555060208201518160000160046101000a8154817bffff
ffffffffffffffffffffffffffffffffffffffffffffffffffff02191690837bffffffffffffffffffffff
ffffffffffffffffffffffffffffffffff16021790555050505b50935093915050565b6200089b83838362
00097c60201b60201c565b505050565b620008b383838362000a7060201b60201c565b505050565b60007b
ffffffffffffffffffffffffffffffffffffffffffffffffffffffff80168211156200091e576040517f08
c379a000000000000000000000000000000000000000000000000000000000815260040162000915906200
17ec565b60405180910390fd5b819050919050565b600063ffffffff801682111562000974576040517f08
c379a00000000000000000000000000000000000000000000000000000000081526004016200096b906200
1884565b60405180910390fd5b819050919050565b6200098f83838362000abb60201b60201c565b600073
ffffffffffffffffffffffffffffffffffffffff168373ffffffffffffffffffffffffffffffffffffffff
1603620009eb57620009d58262000ac060201b60201c565b620009e562000b2360201b60201c565b62000a
6b565b600073ffffffffffffffffffffffffffffffffffffffff168273ffffffffffffffffffffffffffff
ffffffffffff160362000a475762000a318362000ac060201b60201c565b62000a4162000b2360201b6020
1c565b62000a6a565b62000a588362000ac060201b60201c565b62000a698262000ac060201b60201c565b
5b5b505050565b62000a838383836 2000b4760201b60201c565b62000ab662000a988462000b4c60201b60
201c565b62000aa98462000b4c60201b60201c565b8362000bb560201b60201c565b505050565b50505056
5b62000b20600560008373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffff
ffffffffffffffffffff16815260200190815260200160002062000b148362000dd860201b60201c565b62
000e2060201b60201c565b50565b62000b456006 62000b39620005c460201b60201c565b62000e2060201b
60201c565b565b505050565b6000600c60008373ffffffffffffffffffffffffffffffffffffffff1673ff
ffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060009054906101000a
900473ffffffffffffffffffffffffffffffffffffffff169050919050565b8173ffffffffffffffffffff
ffffffffffffffffffff168373ffffffffffffffffffffffffffffffffffffffff161415801562000bf257
50600081115b1562000dd357600073ffffffffffffffffffffffffffffffffffffffff168373ffffffffff
ffffffffffffffffffffffffffffff161462000ce55760008062000c8c600d60008773ffffffffffffffff
ffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152
```

```
60200160002062000eac60201b620012041785620005e660201b60201c565b915091508473ffffffffffff
ffffffffffffffffffffffffffff167fdec2bacdd2f05b59de34da9b523dff8be42e5e38e818c82fdb0bae
774387a724838360405162000cda929190620018a6565b60405180910390a250505b600073ffffffffffff
ffffffffffffffffffffffffffff168273ffffffffffffffffffffffffffffffffffffffff161462000dd2
5760008062000d79600d60008673ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffff
ffffffffffffffffffffffffff168152602001908152602001600020620005ce60201b620011ee17856200
05e660201b60201c565b915091508373ffffffffffffffffffffffffffffffffffffffff167fdec2bacdd2
f05b59de34da9b523dff8be42e5e38e818c82fdb0bae774387a724838360405162000dc792919062 0018a6
565b60405180910390a250505b5b505050565b60008060008373ffffffffffffffffffffffffffffffffff
ffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600020549050
919050565b600062000e3262000ec460201b60201c565b90508062000e498460000162000edd60201b6020
1c565b101562000ea757826000018190806001815401808255809150506001900390600052602060002001
600090919091909150558260010182908060018154018082558091505060019003906000526020600020 01
60009091909190915055 5b505050565b600081836200 0ebc91906200170c565b905092915050565b600062
000ed8600862000f3060201b60201c565b905090565b600080828054905003620000ef557600090506200 0f
2b565b8160018380549050620000f0991906200170c565b8154811062000f1d5762000f1c62001747565b5b
906000526020600020015490505b919050565b600081600001549050919050565b6000815190509190505 6
5b7f4e487b7100000000000000000000000000000000000000000000000000000000600052604160045260
246000fd5b7f4e487b71000000000000000000000000000000000000000000000000000000006000526022
60045260246000fd5b6000600282049050600182168062000fc057607f821691505b60208210810362000f
d65762000fd562000f78565b5b50919050565b600081905081600052602060002090509190505 65b600060
20601f830104905091905056 5b600082821b905092915050565b60006008830262001040 7fffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffff8262001001565b6200104c868362001001
565b955080198416935080861684179250505093925050505 65b60008190509190505 65b60008190509190
50565b600062001099620010936200108d8462001064565b6200106e565b62001064565b905091905056 5b
600081905091905056 5b620010b58362001078565b620010cd620010c482620010a0565b848454620010 0e
565b825550505050565b600090565b620010e4620010d5565b620010f1818484620010aa565b505050565b
5b8181101562001119576200110d600082620010da565b60018101905062001 0f7565b5050565b601f8211
1562001168576200113281 62000fdc565b6200113d8462000ff1565b8101602085101562001 14d57819050
5b62001165620011 5c8562000ff1565b83018262001 0f6565b50505b505050565b600082821c905092915 0
50565b6000620011 8d600019846008026200116d565b198083169150509291505056 5b60006200 11a88383
6200117a565b915082600202821790509291505056 5b620011c38262000f3e565b67ffffffffffffffff81
1115620011df57620011de62000f49565b5b620011eb82546200 0fa7565b620011f8828285620011 1d565b
600060209050601f8311600181146200123057600084156200121b578287015190505b62001227858262 00
119a565b865550620012975 65b601f1984166200124086 62000fdc565b60005b82811015620012 6a578489
01518255600182019150602085019450602081019050620012 43565b8683101562001 28a57848901516200
1286601f891682620011 7a565b8355505b60016002880201885550505 05b50505050505050565b7f4e487b71
00000000000000000000000000000000000000000000000000000000600052601160045260246000fd5b60
008160011c905091905056 5b6000808291508390505b6001851115620013 2d578086048111156200130557
620013046200129f565b5b6001851615620013155780820291505b80810290506200132585620012ce565b
9450620012e5565b94509492505050565b6000826200134857600190506200141b565b8162001358576000
90506200141b565b816001811462001371576002811462001 37c57620013b2565b60019150506200141b56
5b60ff84111562001391576200139062 00129f565b5b8360020a9150848211156200 13ab57620013aa6200
129f565b5b506200141b565b506020831061013383101660 4e8410600b841016171562001 3ec5782820a90
50838111156200 13e657620013e56200129f565b5b6200141b565b620013fb8484846001620012db565b92
50905081840481111562001415576200141462 00129f565b5b81810290505b9392505050565b600060ff82
169050919050565b60006200143c8262001064565b9150620014498362001422565b9250620014787fffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff848462001336565b9050929150
50565b60006200148d8262001064565b91506200149a8362001064565b925082820262001 4aa8162001064
565b9150828204841483151762 0014c457620014c36200129f565b5b50929150505 65b600081905091905 0
565b620014e081620014cb565b82525050565b620014f18162001064565b82525050565b600073ffffffff
ffffffffffffffffffffffffffffffff8216905091905056 5b60006200152482620014f7565b9050919050
565b62001536816200151 7565b82525050565b600060a0820190506200155360008301886200 14d5565b62
0015626020830187620014d5565b620015716040830186620014d5565b6200158060608301856200 14e656
5b6200158f6080830184620015 2b565b9695505050505050565b600082825260208201905092915050565b
7f4552433230566f7465733a20746f74616c20737570706c79207269736b73206f60008201527f76657266
6c6f77696e6720766f7465730000000000000000000000000000000000602082015250565b600062001608 60
308362001599565b91506200161582620015aa565b604082019050919050565b600060208201905081810 3
```

```
60008301526200163b81620015f9565b9050919050565b7f45524332303a206d696e7420746f2074686520
7a65726f206164647265737300600082015250565b60006200167a601f8362001599565b91506200168782
62001642565b602082019050919050565b600060208201905081810360008301526200166ad816200166b56
5b9050919050565b6000620016c18262001064565b9150620016ce8362001064565b925082820190508082
1115620016e957620016e86200129f565b5b92915050565b600060208201905062001706600083018462 00
14e6565b92915050565b6000620017198262001064565b91506200172683620010645 65b92508282039050
818111156200174157620017406200129f565b5b92915050565b7f4e487b71000000000000000000000000
00000000000000000000000000000000600052603260045260246000fd5b7f53616665436173743a207661
6c756520646f65736e27742066697420696e203260008201527f3234206269747300000000000000000000
000000000000000000000000000000602082015250565b6000620017d4602783620015 99565b9150620017
e18262001776565b604082019050919050565b600060208201905081810360008301526200180781620017
c5565b9050919050565b7f53616665436173743a2076616c756520646f65736e27742066697420696e2033
60008201527f3220626974730000000000000000000000000000000000000000000000000000006020820152
50565b60006200186c6026836200159 9565b915062001879826200180e565b604082019050919050565b60
0060208201905081810360008301526200189f816200185d565b9050919050565b60006040820190506200
18bd6000830185620014e6565b620018cc6020830184620014e6565b9392505050565b60805160a05160c0
5160e05161010051610120516143776200192360003960006115d10152600061161301526000 6115f20152
60006115270152600061157d015260006115a601526143776000f3fe608060405234801561001057600080
fd5b50600436106101e55760003560e01c806379cc67901161010f5780639ab24eb0116100a2578063d505
accf11610071578063d505accf146105de578063dd62ed3e146105fa578063f1127ed81461062a578063f2
fde38b1461065a576101e5565b80639ab24eb014610532578063a457c2d714610562578063a9059cbb1461
0592578063c3cda520146105c2576101e5565b80638e539e8c116100de5780638e539e8c146104aa578063
95d89b41146104da5780639711715a146104f8578063981b24d014610502576101e5565b806379cc679014
61042457806 37ecebe00146104405780638cbf9519146104705780638da5cb5b1461048c576101e5565b80
633a46b1a8116101875780635c19a95c116101565780635c19a95c1461039e5780636fcfff45146103ba57
806370a08231146103ea578063715018a61461041a576101e5565b80633a46b1a8146102f257806342966c
68146103225780634ee2cd7e1461033e578063587cde1e1461036e576101e5565b806323b872dd116101c3
57806323b872dd14610256578063313ce567146102865780633644e515146102a457806339509351146102
c2576101e5565b806306fdde03146101ea578063095ea7b31461020857806318160ddd14610238575b6000
80fd5b6101f2610676565b6040516101ff9190612cc7565b60405180910390f35b61022260048036038101
9061021d9190612d82565b610708565b60405161022f9190612ddd565b60405180910390f35b6102406107
2b565b60405161024d9190612e07565b60405180910390f35b61027060048036038101906102 6b9190612e
22565b610735565b60405161027d9190612ddd565b60405180910390f35b61028e610764565b6040516102
9b9190612e91565b60405180910390f35b6102ac61076d565b6040516102b99190612ec5565b6040518091
0390f35b6102dc60048036038101906102d79190612d82565b61077c565b6040516102e99190612ddd565b
60405180910390f35b61030c60048036038101906103079190612d82565b6107b3565b6040516103199190
612e07565b60405180910390f35b61033c60048036038101906103379190612ee0565b610847565b005b61
03586004803603810190610353 9190612d82565b61085b565b6040516103659190612e07565b6040518091
0390f35b61038860048036038101906103839190612f0d565b6108cb565b6040516103959190612f49565b
60405180910390f35b6103b860048036038101906103b39190612f0d565b610934565b005b6103d4600480
36038101906103cf9190612f0d565b610948565b6040516103e19190612f83565b60405180910390f35b61
040460048036038101906103ff9190612f0d565b61099c565b6040516104119190612e07565b6040518091
0390f35b6104226109e4565b005b61043e60048036038101906104399190612d82565b6109f8565b005b61
045a60048036038101906104559190612f0d565b610a18565b6040516104679190612e07565b6040518091
0390f35b61048a60048036038101906104859190612fca565b610a68565b005b610494610a8d565b604051
6104a19190612f49565b60405180910390f35b6104c460048036038101906104bf9190612ee0565b610ab7
565b6040516104d19190612e07565b60405180910390f35b6104e2610b0d565b6040516104ef9190612cc7
565b60405180910390f35b610500610b9f565b005b61051c60048036038101906105179190612ee0565b61
0bb2565b6040516105299190612e07565b60405180910390f35b61054c6004803603810190610547919061
2f0d565b610be3565b6040516105599190612e07565b60405180910390f35b61057c600480360381019061
05779190612d82565b610cf4565b6040516105899190612ddd565b60405180910390f35b6105ac60048036
038101906105a79190612d82565b610d6b565b6040516105b99190612ddd565b60405180910390f35b6105
dc60048036038101906105d7919061304f565b610d8e565b005b6105f860048036038101906105f3919061
30dc565b610e92565b005b610614600480360381019061060f919061317e565b610fd4565b604051610621
9190612e07565b60405180910390f35b610644600480360381019061063f91906131ea565b61105b565b60
40516106519190613 29f565b60405180910390f35b610674600480360381019061066f9190612f0d565b61
116b565b005b606060038054610685906132e9565b80601f0160208091040260200160405190810160405 2
809291908181526020018280546106b1906132e9565b80156106fe5780601f106106d35761010080835404
```

```
02835291602001916106fe565b82019190600052602060002090 5b81548152906001019060200180831161
06e157829003601f168201915b5050505050905090565b60008061071361121a565b905061072081858561
1222565b600191505092915050565b6000600254905090565b60008061074061121a565b905061074d8582
856113eb565b61075885858561147756 5b600191505093925050505 65b60006012905090565b6000610777
611523565b905090565b60008061078761121a565b90506107a881858561079985896 10fd4565b6107a391
90613349565b611222565b600191505092915050565b60004382106107f7576040517f08c379a000000000
00000000000000000000000000000000000000000000000081526004016107ee906133c9565b6040518091
0390fd5b61083f600d60008573ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffff
ffffffffffffffffffffffff1681526020019081526020016000208361163d565b905092915050565b6108
586108526112 1a565b82611749565b50565b60008060006108a884600560008873ffffffffffffffffffff
ffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526020019081526020
01600020611757565b91509150816108bf576108ba8561099c565b6108c1565b805b925050509291505056
5b6000600c60008373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffff
ffffffffffffffff16815260200190815260200160002060009054906101000a900473ffffffffffffffff
ffffffffffffffffffffffff169050919050565b6109456109 3f61121a565b8261184c565b50565b600061
0995600d60008473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffff
ffffffffffffff168152602001908152602001600020805490506119665 65b9050919050565b6000806000
8373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffff
ff16815260200190815260200160002054905091905056 5b6109ec6119b9565b6109f66000611a37565b56
5b610a0a82610a0461121a565b836113eb565b610a148282611749565b5050565b6000610a61600a600084
73ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff
168152602001908152602001600020611afd565b9050919050565b610a706119b9565b80600f6000610100
0a81548160ff02191690831515021790555050565b600060096000905490610 1000a900473ffffffffffff
ffffffffffffffffffffffffffff16905090565b600043821061 0afb576040517f08c379a0000000000000
000000000000000000000000000000000000000000008152600401610af2906133c9565b60405180910390
fd5b610b06600e8361163d565b9050919050565b6060600480546 10b1c906132e9565b80601f0160208091
0402602001604051908101604052809291908181526020018280546 10b48906132e9565b8015610b955780
601f10610b6a5761010080835404028352916020019161 0b95565b820191906000526020600020905b8154
81529060010190602001808311610b7857829003601f168201915b5050505050905090565b610ba76119b9
565b610baf611b0b565b50565b6000806000610bc2846006611757565b915091508161 0bd857610bd36107
2b565b610bda565b805b9250505091905056 5b600080600d60008473ffffffffffffffffffffffffffffff
ffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002080
54905090506000811461 0ccb57600d60008473ffffffffffffffffffffffffffffffffffffffff1673ffff
ffffffffffffffffffffffffffffffff16815260200190815260200160002060018261 0c7f91906133
e9565b81548110610c9057610c8f61341d565b5b9060005260206000200160000160049054906101000a90
047bffffffffffffffffffffffffffffffffffffffffffffffffffffffff16610cce565b60005b7bffffff
ffffffffffffffffffffffffffffffffffffffffffffffffff169150509190505 65b600080610cff61121a
565b90506000610d0d8286610fd4565b905083811015610d52576040517f08c379a0000000000000000000
000000000000000000000000000000000000008152600401610d49906134be565b60405180910390fd5b61
0d5f8286868403611222565b600192505050929150505 65b600080610d7661121a565b9050610d83818585
611477565b600191505092915050565b83421115610dd1576040517f08c379a00000000000000000000000
00000000000000000000000000000000008152600401610dc89061352a565b60405180910390fd5b600061
0e33610e2b7fe48329057bfd03d55e49b547132e39cffd9c1820ad7b9d4c5307691425d15adf8989896040
51602001610e10949392919061354a565b6040516020818303038152906040528051906020012 0611b6156
5b858585611b7b565b9050610e3e81611ba6565b8614610e7f576040517f08c379a0000000000000000000
000000000000000000000000000000000000008152600401610e76906135db565b60405180910390fd5b61
0e89818861184c565b50505050505050565b83421115610ed5576040517f08c379a0000000000000000000
000000000000000000000000000000000000008152600401610ecc90613647565b60405180910390fd5b60
007f6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64845d6126c988888861 0f048c611b
a6565b89604051602001610f1a969594939291906136 67565b6040516020818303038152906040528051 90
602001209050600061 0f3d82611b61565b90506000610f4d82878787611b7b565b90508973ffffffffffff
ffffffffffffffffffffffffffff168173ffffffffffffffffffffffffffffffffffffffff1614610fbd57
6040517f08c379a0000000000000000000000000000000000000000000000000000000008152600401610f
b490613714565b60405180910390fd5b610fc88a8a8a611222565b5050505050505050505050505050565b60006001
60008473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffff
ffffff16815260200190815260200160002060008373ffffffffffffffffffffffffffffffffffffffff16
73ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002054905092915050
565b611063612bf9565b600d60008473ffffffffffffffffffffffffffffffffffffffff1673ffffffffff
```

```
ffffffffffffffffffffffffffffff1681526020019081526020016000208263ffffffff16815481106110
ba576110b961341d565b5b9060005260206000200160405180604001604052908160008201600090549061
01000a900463ffffffff1663ffffffff1663ffffffff1681526020016000820160049054906101000a9004
7bffffffffffffffffffffffffffffffffffffffffffffffffffffffff167bffffffffffffffffffffffff
ffffffffffffffffffffffffffffffff167bffffffffffffffffffffffffffffffffffffffffffffffffff
ffffff1681525050905092915050565b6111736119b9565b600073ffffffffffffffffffffffffffffffff
ffffffff168173ffffffffffffffffffffffffffffffffffffffff16036111e2576040517f08c379a00000
000000000000000000000000000000000000000000000000000081526004016111d9906137a6565b604051
80910390fd5b6111eb81611a37565b50565b600081836111fc9190613349565b905092915050565b600081
8361121291906133e9565b905092915050565b600033905090565b600073ffffffffffffffffffffffffff
ffffffffffffff168373ffffffffffffffffffffffffffffffffffffffff1603611291576040517f08c379
a000000000000000000000000000000000000000000000000000000000815260040161128890613838565b
60405180910390fd5b600073ffffffffffffffffffffffffffffffffffffffff168273ffffffffffffffff
ffffffffffffffffffffffff1603611300576040517f08c379a00000000000000000000000000000000000
000000000000000000000081526004016112f7906138ca565b60405180910390fd5b80600160008573ffff
ffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152
60200190815260200160002060008473ffffffffffffffffffffffffffffffffffffffff1673ffffffffff
ffffffffffffffffffffffffffffff168152602001908152602001600020819055508173ffffffffffffff
ffffffffffffffffffffffffff168373ffffffffffffffffffffffffffffffffffffffff167f8c5be1e5eb
ec7d5bd14f71427d1e84f3dd0314c0f7b2291e5b200ac8c7c3b925836040516113de9190612e07565b6040
5180910390a3505050565b60006113f78484610fd4565b90507fffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffff81146114715781811015611463576040517f08c379a0000000000000
00000000000000000000000000000000000000000000815260040161145a90613936565b60405180910390
fd5b6114708484848403611222565b5b50505050565b6000600f60009054906101000a900460ff16156114
b457606460028361149d9190613956565b6114a791906139c7565b90506114b38482611749565b5b600081
836114c291906133e9565b905081816114d09190613349565b8314611511576040517f08c379a000000000
000000000000000000000000000000000000000000000000815260040161150890613a44565b6040518091
0390fd5b61151c858583611c04565b5050505050565b60007f000000000000000000000000000000000000
000000000000000000000000000073ffffffffffffffffffffffffffffffffffffffff163073ffffffffff
ffffffffffffffffffffffffffffff1614801561159f57507f000000000000000000000000000000000000
000000000000000000000000000046145b156115cc577f0000000000000000000000000000000000000000
000000000000000000000000905061163a565b6116377f0000000000000000000000000000000000000000
0000000000000000000000007f000000000000000000000000000000000000000000000000000000000000
00007f0000000000000000000000000000000000000000000000000000000000000000611e83565b90505b
90565b6000808380549050905060005b818110156116bc57600061165e8284611ebd565b90508486828154
8110611674576116736134 1d565b5b9060005260206000200160000160009054906101000a900463ffffff
ff1663ffffffff1611156116a6578092506116b6565b6001816116b391906133 49565b91505b5061164a56
5b6000821461171e57846001836116d291906133e9565b815481106116e3576116e261341d565b5b906000
5260206000200160000160049054906101000a90047bffffffffffffffffffffffffffffffffffffffffff
ffffffffffffff16611721565b60005b7bffffffffffffffffffffffffffffffffffffffffffffffffffff
ffff1692505050929150505 65b6117538282611ee3565b5050565b600080600084 1161179d576040517f08
c379a000000000000000000000000000000000000000000000000000000000815260040161179490613ab0
565b60405180910390fd5b6117a5611f01565b8411156117e7576040517f08c379a0000000000000000000
0000000000000000000000000000000000000081526004016117de90613b1c565b60405180910390fd5b60
006117ff8585600001611f1290919063ffffffff16565b90508360000180549050810361181c5760008092
5092505061184556 5b6001846001018281548110611834576118336134 1d565b5b90600052602060002001
5492509250505b9250929050565b6000611857836108cb565b905060006118648461099c565b905082600c
60008673ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffff
ffffff16815260200190815260200160002060006101000a81548173ffffffffffffffffffffffffffffff
ffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff1602179055508273ffffffff
ffffffffffffffffffffffffffffffff168273ffffffffffffffffffffffffffffffffffffffff168573ff
ffffffffffffffffffffffffffffffffffffff167f3134e8a2e6d97e929a7e54011ea5485d7d196dd5f0ba
4d4ef95803e8e3fc257f60405160405180910390a4611960828483611feb565b50505050565b600063ffff
ffff80168211156119b1576040517f08c379a0000000000000000000000000000000000000000000000000
0000000081526004016119a890613bae565b60405180910390fd5b81905091905056 5b6119c161121a565b
73ffffffffffffffffffffffffffffffffffffffff166119df610a8d565b73ffffffffffffffffffffffff
ffffffffffffffff1614611a35576040517f08c379a0000000000000000000000000000000000000000000
000000000000008152600401611a2c90613c1a565b60405180910390fd5b565b6000600960009054906101
```

```
000a900473ffffffffffffffffffffffffffffffffffffffff16905081600960006101000a81548173ffff
ffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffff
ff1602179055508173ffffffffffffffffffffffffffffffffffffffff168173ffffffffffffffffffffff
ffffffffffffffffff167f8be0079c531659141344cd1fd0a4f28419497f9722a3daafe3b4186f6b6457e0
60405160405180910390a35050565b600081600001549050919050565b6000611b1760086121e4565b6000
611b21611f01565b90507f8030e83b04d87bef53480e26263266d6ca66863aa8506aca6f2559d18aa1cb67
81604051611b529190612e07565b60405180910390a18091505090565b6000611b74611b6e611523565b83
6121fa565b9050919050565b6000806000611b8c8787878761222d565b91509150611b9981612339565b81
92505050949350505050565b600080600a60008473ffffffffffffffffffffffffffffffffffffffff1673
ffffffffffffffffffffffffffffffffffffffff1681526020019081526020016000209050611bf381611a
fd565b9150611bfe816121e4565b50919050565b600073ffffffffffffffffffffffffffffffffffffffff
168373ffffffffffffffffffffffffffffffffffffffff1603611c73576040517f08c379a0000000000000
000000000000000000000000000000000000000000008152600401611c6a90613cac565b60405180910390
fd5b600073ffffffffffffffffffffffffffffffffffffffff168273ffffffffffffffffffffffffffffff
ffffffffff1603611ce2576040517f08c379a0000000000000000000000000000000000000000000000000
000000008152600401611cd990613d3e565b60405180910390fd5b611ced838383612505565b6000806000
8573ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffff
ff16815260200190815260200160002054905081811015611d73576040517f08c379a00000000000000000
00000000000000000000000000000000000000008152600401611d6a90613dd0565b60405180910390fd5b
8181036000808673ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffff
ffffffffffffff16815260200190815260200160002081905550816000808573ffffffffffffffffffffff
ffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001
6000206000828254611e069190613349565b925050819055508273ffffffffffffffffffffffffffffffff
ffffffff168473ffffffffffffffffffffffffffffffffffffffff167fddf252ad1be2c89b69c2b068fc37
8daa952ba7f163c4a11628f55a4df523b3ef84604051611e6a9190612e07565b60405180910390a3611e7d
848484612515565b50505050565b6000838383463060405160200161 1e9e959493929190613df0565b6040
51602081830303815290604052805190602001209050939250505050565b60006002828418611ece91906139
c7565b828416611edb9190613349565b905092915050565b611eed8282612525565b611efb600e61120483
6126fb565b50505050565b6000611f0d6008611afd565b905090565b60008083805490500361 1f28576000
9050611fe5565b600080848054905090505b8082101561 1f8c576000611f478383611ebd565b9050848682
81548110611f5d57611f5c61341d565b5b90600052602060002001541115611f7657809150611f86565b60
0181611f839190613349565b92505b50611f33565b600082118015611fc457508385600184611fa6919061
33e9565b81548110611fb757611fb661341d565b5b906000526020600020015414 5b15611fdf5760018261
1fd691906133e9565b92505050611fe5565b81925050505b92915050565b8173ffffffffffffffffffffff
ffffffffffffffffff168373ffffffffffffffffffffffffffffffffffffffff1614158015612027575060
0081115b156121df57600073ffffffffffffffffffffffffffffffffffffffff168373ffffffffffffffff
ffffffffffffffffffffffff1614612105576000806120ae600d60008773ffffffffffffffffffffffffff
ffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526020019081526020016000
20611204856126fb565b915091508473ffffffffffffffffffffffffffffffffffffffff167fdec2bacdd2
f05b59de34da9b523dff8be42e5e38e818c82fdb0bae774387a72483836040516120fa929190613e43565b
60405180910390a250505b600073ffffffffffffffffffffffffffffffffffffffff168273ffffffffffff
ffffffffffffffffffffffffffff16146121de57600080612187600d60008673ffffffffffffffffffffff
ffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001
6000206111ee856126fb565b915091508373ffffffffffffffffffffffffffffffffffffffff167fdec2ba
cdd2f05b59de34da9b523dff8be42e5e38e818c82fdb0bae774387a72483836040516121d3929190613e43
565b60405180910390a250505b5b505050565b600181600001600082825401925050819055505050565b6000
828260405160200161220f929190613ee4565b6040516020818303038152906040528051906020012090 50
92915050565b6000807f7fffffffffffffffffffffffffffffff5d576e7357a4501ddfe92f46681b20a083
60001c111561226857600060039150915061233056 5b601b8560ff1614158015612280575060 1c8560ff16
14155b15612292576000600491509150612330565b60006001878787876040516000815260200160405260
40516122b79493929190613f1b565b6020604051602081039080840390855afa158015 6122d9573d600080
3e3d6000fd5b505050602060405103519050600073ffffffffffffffffffffffffffffffffffffffff1681
73ffffffffffffffffffffffffffffffffffffffff16036123275760006001925092505061 2330565b8060
0092509250505b94509492505050565b600060048111156123 4d5761234c613f60565b5b81600481111561
23605761235f613f60565b5b0315612502576001600481111561237a57612379613f60565b5b8160048111
1561238d5761238c613f60565b5b036123cd576040517f08c379a000000000000000000000000000000000
0000000000000000000000008152600401 6123c490613fdb565b60405180910390fd5b600260048111 1561
23e1576123e0613f60565b5b81600481111561 23f4576123f3613f60565b5b0361243457 6040517f08c379
```

```
a000000000000000000000000000000000000000000000000000000000815260040161242b90614047565b
60405180910390fd5b6003600481111561244857612447613f60565b5b81600481111561245b5761245a61
3f60565b5b0361249b576040517f08c379a000000000000000000000000000000000000000000000000000
00000081526004016124929061 40d9565b60405180910390fd5b6004808111156124ae576124ad613f6056
5b5b8160048111156124c1576124c0613f60565b5b03612501576040517f08c379a0000000000000000000
0000000000000000000000000000000000000081526004016124f89061416b565b60405180910390fd5b5b
50565b612510838383612973565b505050565b612520838383612a2b565b505050565b600073ffffffffff
ffffffffffffffffffffffffffffff168273ffffffffffffffffffffffffffffffffffffffff1603612594
576040517f08c379a000000000000000000000000000000000000000000000000000000000815260040161
258b906141fd565b60405180910390fd5b6125a082600083612505565b60008060008473ffffffffffffff
ffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526020019081
5260200160002054905081811015612626576040517f08c379a00000000000000000000000000000000000
000000000000000000000081526004016 1261d9061428f565b60405180910390fd5b8181036000808573ff
ffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681
52602001908152602001600020819055508160026000828254 61267d91906133e9565b9250508190555060
0073ffffffffffffffffffffffffffffffffffffffff168373ffffffffffffffffffffffffffffffffffff
ffff167fddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef846040516126e2
9190612e07565b60405180910390a36126f683600084612515565b505050565b6000806000858054905090
5060008114612769578560018261271d91906133e9565b8154811061272e5761272d61341d565b5b906000
5260206000200160000160049054906101000a90047bffffffffffffffffffffffffffffffffffffffffff
ffffffffffffff1661276c565b60005b7bffffffffffffffffffffffffffffffffffffffffffffffffffff
ffff16925061279a83858763ffffffff16565b915060008111801561 27ed575043866001836127b6919061
33e9565b815481106127c7576127c661341d565b5b9060005260206000200160000160009054906101000a
900463ffffffff1663ffffffff16145b1561287a576127fb82612a56565b8660018361280991906133e956
5b8154811061281a5761281961341d565b5b9060005260206000200160000160046101000a8154817bffff
ffffffffffffffffffffffffffffffffffffffffffffffffffff02191690837bffffffffffffffffffffff
ffffffffffffffffffffffffffffffffff16021790555061296a565b85604051806040016040528061288f
43611966565b63ffffffff168152602001 6128a385612a56565b7bffffffffffffffffffffffffffffffff
ffffffffffffffffffffffff16815250908060018154018082558091505060019003906000526020600020
0160009091909190915060008201518160000160006101000a81548163ffffffff02191690836 3ffffffff
16021790555060208201518160000160046101000a8154817bffffffffffffffffffffffffffffffffffff
ffffffffffffffffffff02191690837bffffffffffffffffffffffffffffffffffffffffffffffffffffff
ff16021790555050505b509350939150505 65b61297e838383612ac1565b600073ffffffffffffffffffff
ffffffffffffffffffff168373ffffffffffffffffffffffffffffffffffffffff16036129c8576129bb82
612ac6565b6129c3612b19565b612a26565b600073ffffffffffffffffffffffffffffffffffffffff1682
73ffffffffffffffffffffffffffffffffffffffff1603612a1257612a0583612ac6565b612a0d612b1956
5b612a25565b612a1b83612ac6565b612a2482612ac6565b5b5b505050565b612a36838383612b2d565b61
2a51612a42846108cb565b612a4b846108cb565b83611feb565b505050565b60007bffffffffffffffffff
ffffffffffffffffffffffffffffffffffffff8016821115612ab9576040517f08c379a000000000000000
0000000000000000000000000000000000000000008152600401612ab090614321565b60405180910390fd
5b819050919050565b505050565b612b16600560008373ffffffffffffffffffffffffffffffffffffffff
1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002 0612b11836109
9c565b612b32565b50565b612b2b6006612b2661072b565b612b32565b565b505050565b6000612b3c611f
01565b905080612b4b84600001612bad565b1015612ba85782600001819080600181540180825580915050
6001900390600052602060002001600090919091909150558260010182908060018154018082558091505 0
6001900390600052602060002001600090919091909150555b505050565b600080828054905003612bc357
60009050612bf4565b8160018380549050612bd591906133e9565b81548110612be657612be561341d565b
5b906000526020600020015490505b919050565b60405180604001604052806000 63ffffffff1681526020
0160007bffffffffffffffffffffffffffffffffffffffffffffffffffffffff1681525090565b60008151
9050919050565b600082825260208201905092915050565b60005b83811015612c715780820151818401 52
602081019050612c56565b60008484015250505050565b6000601f19601f8301169050919050565b600061
2c9982612c37565b612ca38185612c42565b9350612cb38185602086016 12c53565b612cbc81612c7d565b
840191505092915050565b60006020820190508181036000830152 612ce18184612c8e565b905092915050
565b600080fd5b600073ffffffffffffffffffffffffffffffffffffffff8216905091905056 5b6000612d
1982612cee565b9050919050565b612d2981612d0e565b8114612d3457600080fd5b50565b600081359050
612d4681612d20565b92915050565b6000819050919050565b612d5f81612d4c565b8114612d6a57600080
fd5b50565b600081359050612d7c81612d56565b92915050565b60008060408385031215612d9957612d98
612ce9565b5b6000612da785828601612d37565b9250506020612db885828601612d6d565b915050925092
```

```
9050565b60008115159050919050565b612dd781612dc2565b82525050565b6000602082019050612df260
00830184612dce565b92915050565b612e0181612d4c565b82525050565b6000602082019050612e1c6000
830184612df8565b92915050565b600080600060608486031215612e3b57612e3a612ce9565b5b6000612e
4986828701612d37565b9350506020612e5a86828701612d37565b9250506040612e6b86828701612d6d56
5b9150509250925092565b600060ff82169050919050565b612e8b81612e75565b82525050565b60006020
82019050612ea66000830184612e82565b92915050565b6000819050919050565b612ebf81612eac565b82
525050565b6000602082019050612eda6000830184612eb6565b92915050565b600060208284031215612e
f657612ef5612ce9565b5b6000612f0484828501612d6d565b91505092915050565b600060208284031215
612f2357612f22612ce9565b5b6000612f3184828501612d37565b91505092915050565b612f4381612d0e
565b82525050565b6000602082019050612f5e6000830184612f3a565b92915050565b600063ffffffff82
169050919050565b612f7d81612f64565b82525050565b6000602082019050612f986000830184612f7456
5b92915050565b612fa781612dc2565b8114612fb257600080fd5b50565b600081359050612fc481612f9e
565b92915050565b600060208284031215612fe057612fdf612ce9565b5b6000612fee84828501612fb556
5b91505092915050565b61300081612e75565b811461300b57600080fd5b50565b60008135905061301d81
612ff7565b92915050565b61302c81612eac565b811461303757600080fd5b50565b600081359050613049
81613023565b92915050565b60008060008060008060c0878903121561306c5761306b612ce9565b5b6000
61307a89828a01612d37565b965050602061308b89828a01612d6d565b955050604061309c89828a01612d
6d565b94505060606130ad89828a0161300e565b93505060806130be89828a0161303a565b92505060a061
30cf89828a0161303a565b9150509295509295509295565b600080600080600080600060e0888a03121561
30fb576130fa612ce9565b5b60006131098a828b01612d37565b975050602061311a8a828b01612d37565b
965050604061312b8a828b01612d6d565b955050606061313c8a828b01612d6d565b945050608061314d8a
828b0161300e565b93505060a061315e8a828b0161303a565b92505060c061316f8a828b0161303a565b91
50509295989194975092955050565b6000806040838503121561319557613194612ce9565b5b60006131a385
828601612d37565b92505060206131b485828601612d37565b9150509250929050565b6131c781612f6456
5b81146131d257600080fd5b50565b6000813590506131e4816131be565b92915050565b60008060408385
03121561320157613200612ce9565b5b600061320f85828601612d37565b925050602061322085828601 61
31d5565b9150509250929050565b61323381612f64565b82525050565b60007bffffffffffffffffffffff
ffffffffffffffffffffffffffffffffff82169050919050565b61326a81613239565b82525050565b6040
8201600082015161328660008501826 1322a565b5060208201516132996020850182613261565b50505050
565b60006040820190506132b46000830184613270565b92915050565b7f4e487b71000000000000000000
00000000000000000000000000000000000000600052602260045260246000fd5b60006002820490506001
82168061330157607f821691505b602082108103613314576133136132ba565b5b50919050565b7f4e487b
7100000000000000000000000000000000000000000000000000000000600052601160045260246000fd5b
60006133548 2612d4c565b915061335f83612d4c565b92508282019050808211156133775761337661331a
565b5b92915050565b7f4552433230566f7465733a20626c6f636b206e6f7420796574206d696e65640060
0082015250565b60006133b3601f83612c42565b91506133be8261337d565b602082019050919050565b60
0060208201905081810360008301526133e2816133a6565b9050919050565b60006133f482612d4c565b91
506133ff83612d4c565b925082820390508181111561341757613416 61331a565b5b92915050565b7f4e48
7b7100000000000000000000000000000000000000000000000000000000600052603260045260246000fd
5b7f45524332303a2064656372656173656420616c6c6f77616e63652062656c6f7760008201527f207a65
726f000000000000000000000000000000000000000000000000000000602082015250565b60006134a860
2583612c42565b91506134b38261344c565b602082019050919050565b6000602082019050818103600083
01526134d78161349b565b9050919050565b7f4552433230566f7465733a207369676e6174757265206578
7069726564000000600082015250565b6000613514601d83612c42565b915061351f826134de565b602082
019050919050565b600060208201905081810360008301526135438161350756 5b9050919050565b600060
808201905061355f6000830187612eb6565b61356c6020830186612f3a565b6135796040830185612df856
5b6135866060830184612df8565b95945050505050565b7f4552433230566f7465733a20696e76616c6964
206e6f6e63650000000000000000600082015250565b60006135c5601983612c42565b91506135d08261358f
565b602082019050919050565b600060208201905081810360008301526135f4816135b8565b9050919050
565b7f45524332305065726d69743a206578706972656420646561646c696e65000000600082015250565b
6000613631601d83612c42565b915061363c826135fb565b602082019050919050565b6000602082019050
818103600083015261366081613624565b9050919050565b600060c08201905061367c6000830189612eb6
565b6136896020830188612f3a565b6136966040830187612f3a565b6136a36060830186612df8565b6136
b06080830185612df8565b6136bd60a0830184612df8565b979650505050505050565b7f45524332305065
726d69743a20696e76616c6964207369676e617475726500006000820152 50565b60006136fe601e83612c
42565b9150613709826136c8565b602082019050919050565b600060208201905081810360008301526137
2d816136f1565b9050919050565b7f4f776e61626c653a206e6577206f776e657220697320746865207a65
726f206160008201527f6464726573730000000000000000000000000000000000000000000000000000060
```

```
2082015250565b6000613790602683612c42565b915061379b82613734565b604082019050919050565b60
006020820190508181036000830152 6137bf81613783565b9050919050565b7f45524332303a2061707072
6f76652066726f6d20746865207a65726f2061646460008201527f72657373000000000000000000000000
00000000000000000000000000000000602082015250565b6000613822602483612c42565b915061382d82
6137c6565b604082019050919050565b6000602082019050818103600083015261385181613815565b9050
919050565b7f45524332303a20617070726f766520746f20746865207a65726f20616464726560008201 52
7f7373000000000000000000000000000000000000000000000000000000000000602082015250565b6000
6138b4602283612c42565b91506138bf82613858565b604082019050919050565b60006020820190508181
036000830152 6138e3816138a7565b9050919050565b7f45524332303a20696e73756666696369656e7420
616c6c6f77616e6365000000600082015250565b6000613920601d83612c42565b915061392b826138ea56
5b602082019050919050565b6000602082019050818103600083015261394f81613913565b905091905056
5b60006139618261 2d4c565b915061396c83612d4c565b925082820261397a81612d4c565b915082820484
148315176139915761399061331a565b5b5092915050565b7f4e487b710000000000000000000000000000
0000000000000000000000000000600052601260045260246000fd5b60006139d282612d4c565b91506139
dd83612d4c565b9250826139ed576139ec613998565b5b828204905092915050565b7f4275726e2076616c
756520696e76616c69640000000000000000000000000000600082015250565b6000613a2e601283612c42
565b9150613a39826139f8565b602082019050919050565b6000602082019050818103600083015 2613a5d
81613a21565b9050919050565b7f4552433230536e617073686f743a20696420697320300000000000000
0000006000820152 50565b6000613a9a601683612c42565b9150613aa582613a64565b6020820190509190
50565b6000602082019050818103600083015 2613ac981613a8d565b9050919050565b7f4552433230536e
617073686f743a206e6f6e6578697374656e74206964000000600082015250565b6000613b06601d83612c
42565b9150613b1182613ad0565b602082019050919050565b6000602082019050818103600083015 2613b
3581613af9565b9050919050565b7f53616665436173743a2076616c756520646f65736e27742066697420
696e203360008201527f3220626974730000000000000000000000000000000000000000000000000060
2082015250565b6000613b98602683612c42565b9150613ba382613b3c565b604082019050919050565b60
0060208201905081810360008301 52613bc781613b8b565b9050919050565b7f4f776e61626c653a206361
6c6c6572206973206e6f7420746865206f776e6572600082015250565b6000613c04602083612c42565b91
50613c0f82613bce565b602082019050919050565b6000602082019050818103600083015 2613c3381613b
f7565b9050919050565b7f45524332303a207472616e736665722066726f6d20746865207a65726f206164
60008201527f64726573730000000000000000000000000000000000000000000000000000006020820152
50565b6000613c96602583612c42565b9150613ca182613c3a565b604082019050919050565b6000602082
0190508181036000830152613cc581613c89565b9050919050565b7f45524332303a207472616e73666572
20746f20746865207a65726f206164647260008201527f6573730000000000000000000000000000000000
00000000000000000000000000602082015250565b6000613d28602383612c42565b9150613d3382613ccc56
5b604082019050919050565b6000602082019050818103600083015 2613d5781613d1b565b905091905056
5b7f45524332303a207472616e7366657220616d6f756e7420657863656564732062 60008201527f616c61
6e63650000000000000000000000000000000000000000000000000000602082015250565b6000613dba60
2683612c42565b9150613dc582613d5e565b604082019050919050565b600060208201905081810360008 3
0152613de981613dad565b9050919050565b600060a082019050613e0560008301 88612eb6565b613e1260
20830187612eb6565b613e1f6040830186612eb6565b613e2c6060830185612df8565b613e396080830184
612f3a565b9695505050505050565b600060408201905061 3e586000830185612df8565b613e6560208301
84612df8565b9392505050565b600081905092915050565b7f190100000000000000000000000000000000
00000000000000000000000000006000820152 50565b6000613ead600283613e6c565b9150613eb882613e
77565b600282019050919050565b6000819050919050565b613ede613ed982612eac565b613ec3565b8252
5050565b6000613eef82613ea0565b9150613efb8285613ecd565b602082019150613f0b8284613ecd565b
6020820191508190509392505050565b600060808201905061 3f306000830187612eb6565b613f3d602083
0186612e82565b613f4a6040830185612eb6565b613f576060830184612eb6565b95945050505050565b7f
4e487b7100000000000000000000000000000000000000000000000000000000600052602160045260 2460
00fd5b7f45434453413a20696e76616c6964207369676e617475726500000000000000006000820152 5056
5b6000613fc5601883612c42565b9150613fd082613f8f565b602082019050919050565b60006020820190
508181036000830152613ff481613fb8565b9050919050565b7f45434453413a20696e76616c6964207369
676e6174757265206c656e67746800600082015250565b600061403160 1f83612c42565b915061403c8261
3ffb565b602082019050919050565b6000602082019050818103600083015261406081614024565b905091
9050565b7f45434453413a20696e76616c6964207369676e6174757265202773272076616c60008201527f
756500000000000000000000000000000000000000000000000000000000000602082015250565b600061
40c3602283612c42565b91506140ce82614067565b604082019050919050565b6000602082019050818103
6000830152 6140f2816140b6565b9050919050565b7f45434453413a20696e76616c6964207369676e6174
757265202776272076616c60008201527f7565000000000000000000000000000000000000000000000000
```

```
000000000000602082015250565b6000614155602283612c42565b9150614160826140f9565b6040820190
50919050565b6000602082019050818103600083015261418481614148565b9050919050565b7f45524332
303a206275726e2066726f6d20746865207a65726f2061646472657360008201527f730000000000000000
0000000000000000000000000000000000000000000000602082015250565b60006141e7602183612c4256
5b91506141f28261418b565b604082019050919050565b600060208201905081810360008301526142168161
6141da565b9050919050565b7f45524332303a206275726e20616d6f756e7420657863656564732062616c
616e60008201527f6365000000000000000000000000000000000000000000000000000000000000602082
015250565b6000614279602283612c42565b91506142848261421d565b604082019050919050565b600060
20820190508181036000830152614 2a88161426c565b9050919050565b7f53616665436173743a2076616c
756520646f65736e27742066697420696e20326000820152 7f32342062697473000000000000000000000000
0000000000000000000000000000602082015250565b600061430b602783612c42565b9150614316826142
af565b604082019050919050565b60006020820190508181036000830152614 33a816142fe565b90509190
5056fea26469706673582212208e7c6d4a2698e3af10e4a7574293cac373d423a237dc85a62ea41f604ef4
e03a64736f6c63430008130033
```