```solidity
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.8.0) (proxy/utils/UUPSUpgradeable.sol)

pragma solidity ^0.8.0;

import "../../interfaces/draft-IERC1822.sol";
import "../ERC1967/ERC1967Upgrade.sol";

/**
 * @dev An upgradeability mechanism designed for UUPS proxies. The functions included here can perform an upgrade of an
 * {ERC1967Proxy}, when this contract is set as the implementation behind such a proxy.
 *
 * A security mechanism ensures that an upgrade does not turn off upgradeability accidentally, although this risk is
 * reinstated if the upgrade retains upgradeability but removes the security mechanism, e.g. by replacing
 * `UUPSUpgradeable` with a custom implementation of upgrades.
 *
 * The {_authorizeUpgrade} function must be overridden to include access restriction to the upgrade mechanism.
 *
 * _Available since v4.1._
 */
abstract contract UUPSUpgradeable is IERC1822Proxiable, ERC1967Upgrade {
    /// @custom:oz-upgrades-unsafe-allow state-variable-immutable state-variable-assignment
    address private immutable __self = address(this);

    /**
     * @dev Check that the execution is being performed through a delegatecall call and that the execution context is
     * a proxy contract with an implementation (as defined in ERC1967) pointing to self. This should only be the case
     * for UUPS and transparent proxies that are using the current contract as their implementation. Execution of a
     * function through ERC1167 minimal proxies (clones) would not normally pass this test, but is not guaranteed to
     * fail.
     */
    modifier onlyProxy() {
        require(address(this) != __self, "Function must be called through delegatecall");
        require(_getImplementation() == __self, "Function must be called through active proxy");
        _;
    }

    /**
     * @dev Check that the execution is not being performed through a delegate call. This allows a function to be
     * callable on the implementing contract but not through proxies.
     */
    modifier notDelegated() {
        require(address(this) == __self, "UUPSUpgradeable: must not be called through delegatecall");
        _;
    }

    /**
     * @dev Implementation of the ERC1822 {proxiableUUID} function. This returns the storage slot used by the
     * implementation. It is used to validate the implementation's compatibility when performing an upgrade.
     *
     * IMPORTANT: A proxy pointing at a proxiable contract should not be considered proxiable itself, because this risks
     * bricking a proxy that upgrades to it, by delegating to itself until out of gas. Thus it is critical that this
     * function revert if invoked through a proxy. This is guaranteed by the `notDelegated` modifier.
     */
    function proxiableUUID() external view virtual override notDelegated returns (bytes32) {
        return _IMPLEMENTATION_SLOT;
    }

    /**
     * @dev Upgrade the implementation of the proxy to `newImplementation`.
     *
     * Calls {_authorizeUpgrade}.
     *
     * Emits an {Upgraded} event.
     */
    function upgradeTo(address newImplementation) external virtual onlyProxy {
        _authorizeUpgrade(newImplementation);
        _upgradeToAndCallUUPS(newImplementation, new bytes(0), false);
    }

    /**
     * @dev Upgrade the implementation of the proxy to `newImplementation`, and subsequently execute the function call
     * encoded in `data`.
     *
     * Calls {_authorizeUpgrade}.
     *
     * Emits an {Upgraded} event.
     */
    function upgradeToAndCall(address newImplementation, bytes memory data) external payable virtual onlyProxy {
        _authorizeUpgrade(newImplementation);
        _upgradeToAndCallUUPS(newImplementation, data, true);
    }

    /**
     * @dev Function that should revert when `msg.sender` is not authorized to upgrade the contract. Called by
     * {upgradeTo} and {upgradeToAndCall}.
     *
     * Normally, this function will use an xref:access.adoc[access control] modifier such as {Ownable-onlyOwner}.
     *
     * ```solidity
     * function _authorizeUpgrade(address) internal override onlyOwner {}
     * ```
     */
    function _authorizeUpgrade(address newImplementation) internal virtual;
}
```