```
pragma solidity =0.5.16;

import './interfaces/IUniswapV2ERC20.sol';
import './libraries/SafeMath.sol';

contract UniswapV2ERC20 is IUniswapV2ERC20 {
    using SafeMath for uint;

    string public constant name = 'Uniswap V2';
    string public constant symbol = 'UNI-V2';
    uint8 public constant decimals = 18;
    uint  public totalSupply;
    mapping(address => uint) public balanceOf;
    mapping(address => mapping(address => uint)) public allowance;

    bytes32 public DOMAIN_SEPARATOR;
    // keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)");
    bytes32 public constant PERMIT_TYPEHASH = 0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64845d6126c9;
    mapping(address => uint) public nonces;

    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    constructor() public {
        uint chainId;
        assembly {
            chainId := chainid
        }
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256('EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)'),
                keccak256(bytes(name)),
                keccak256(bytes('1')),
                chainId,
                address(this)
            )
        );
    }

    function _mint(address to, uint value) internal {
        totalSupply = totalSupply.add(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(address(0), to, value);
    }

    function _burn(address from, uint value) internal {
        balanceOf[from] = balanceOf[from].sub(value);
        totalSupply = totalSupply.sub(value);
        emit Transfer(from, address(0), value);
    }

    function _approve(address owner, address spender, uint value) private {
        allowance[owner][spender] = value;
        emit Approval(owner, spender, value);
    }

    function _transfer(address from, address to, uint value) private {
        balanceOf[from] = balanceOf[from].sub(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(from, to, value);
    }

    function approve(address spender, uint value) external returns (bool) {
        _approve(msg.sender, spender, value);
        return true;
    }

    function transfer(address to, uint value) external returns (bool) {
        _transfer(msg.sender, to, value);
        return true;
```

```
71      }
72
73      function transferFrom(address from, address to, uint value) external returns (bool
        ) {
74          if (allowance[from][msg.sender] != uint(-1)) {
75              allowance[from][msg.sender] = allowance[from][msg.sender].sub(value);
76          }
77          _transfer(from, to, value);
78          return true;
79      }
80
81      function permit(address owner, address spender, uint value, uint deadline, uint8 v
        , bytes32 r, bytes32 s) external {
82          require(deadline >= block.timestamp, 'UniswapV2: EXPIRED');
83          bytes32 digest = keccak256(
84              abi.encodePacked(
85                  '\x19\x01',
86                  DOMAIN_SEPARATOR,
87                  keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value, nonces
                    [owner]++, deadline))
88              )
89          );
90          address recoveredAddress = ecrecover(digest, v, r, s);
91          require(recoveredAddress != address(0) && recoveredAddress == owner,
            'UniswapV2: INVALID_SIGNATURE');
92          _approve(owner, spender, value);
93      }
94  }
95
```