



# Armors Labs

## LFG Swap

### Smart Contract Audit

- LFG Swap Audit Summary
- LFG Swap Audit
  - Document information
    - Audit results
    - Audited target file
  - Vulnerability analysis
    - Vulnerability distribution
    - Summary of audit results
    - Contract file
    - Analysis of audit results
      - Re-Entrancy
      - Arithmetic Over/Under Flows
      - Unexpected Blockchain Currency
      - Delegatecall
      - Default Visibilities
      - Entropy Illusion
      - External Contract Referencing
      - Unsolved TODO comments
      - Short Address/Parameter Attack
      - Unchecked CALL Return Values
      - Race Conditions / Front Running
      - Denial Of Service (DOS)
      - Block Timestamp Manipulation
      - Constructors with Care
      - Unintialised Storage Pointers
      - Floating Points and Numerical Precision
      - tx.origin Authentication
      - Permission restrictions

# LFG Swap Audit Summary

Project name : LFG Swap Contract

Project address: None

Code URL : <https://github.com/LfgSwap/lfg-protocol>

Commit : 1025b7626ae27abb9f7bdadec4dffa6d26dfb4d8

Project target : LFG Swap Contract Audit

Blockchain : ETHW

Test result : PASSED

Audit Info

Audit NO : 0X202209060016

Audit Team : Armors Labs

Audit Proofreading: <https://armors.io/#project-cases>

## LFG Swap Audit

The LFG Swap team asked us to review and audit their LFG Swap contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

## Document information

Name	Auditor	Version	Date
LFG Swap Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2022-09-06

## Audit results

NOTICE:

Minter can issue more LFG tokens indefinitely.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the LFG Swap contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 3 months from the date of output.

Disclaimer

Armors Labs Reports is not and should not be regarded as an "approval" or "disapproval" of any particular project or team. These reports are not and should not be regarded as indicators of the economy or value of any "product" or

"asset" created by any team. Armors do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'...)

Armors Labs Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Armors does not guarantee the safety or functionality of the technology agreed to be analyzed.

Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused. Armors Labs Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

## Audited target file

file	md5
LfgToken.sol	31b18fb01b3ef83ed06a2d66f725fd78
ILfgSwapFactory.sol	d6f10907e51f59793f1209ba0b07484f
ILfgSwapRouter.sol	f47c0a7f452267c2344af9e69ad1f9df
IWETH.sol	3afb1bd75b5f02e0be95eca08a2adf572
ILfgSwapPair.sol	67e664c7b92ed41984e529110e570e40
ISwapMining.sol	5768af346049b69afcaae5411010781d
ILfgToken.sol	ab2902b774f1b20d96c67d3925bd27e7
IERC20LfgSwap.sol	98acf9d08b0d56c3e8874a242923f1a6
LfgSwapFactory.sol	78cdd2681a7de23c792a59cac0165807
LfgSwapRouter.sol	d16df05f86a87c25f23810938e2f534b
LfgSwapPair.sol	31bb12e5cf9ea54d56d8d800d230308c
FixedPoint.sol	d563fe0a0b8d4017c677b864d90b6b80
UQ112x112.sol	6060f387e5a25af7de7a8f01c6f4ba20
LfgSwapLibrary.sol	50ebdaf3ad291de71766d2b8dd585d87
SafaMath.sol	91ba36ad2e6d077b3a84ee30c78927a2
TransferHelper.sol	b312dfef05a1e7fdbade150348591bd0

## Vulnerability analysis

### Vulnerability distribution

vulnerability level	number
Critical severity	0

vulnerability level	number
High severity	0
Medium severity	0
Low severity	0

## Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe
Permission restrictions	safe

## Contract file

LfgToken.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/utils/EnumerableSet.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract LfgToken is ERC20, Ownable {
    uint256 private constant preMineSupply = 200000000 * 1e18;

    using EnumerableSet for EnumerableSet.AddressSet;
```

```

EnumerableSet.AddressSet private _minters;

constructor() public ERC20("LfgSwap Finance Token", "LFG"){
    _mint(msg.sender, preMineSupply);
}

// mint with max supply
function mint(address _to, uint256 _amount) public onlyMinter returns (bool) {
    _mint(_to, _amount);
    return true;
}

function addMinter(address _addMinter) public onlyOwner returns (bool) {
    require(_addMinter != address(0), "JfToken: _addMinter is the zero address");
    return EnumerableSet.add(_minters, _addMinter);
}

function delMinter(address _delMinter) public onlyOwner returns (bool) {
    require(_delMinter != address(0), "JfToken: _delMinter is the zero address");
    return EnumerableSet.remove(_minters, _delMinter);
}

function getMinterLength() public view returns (uint256) {
    return EnumerableSet.length(_minters);
}

function isMinter(address account) public view returns (bool) {
    return EnumerableSet.contains(_minters, account);
}

function getMinter(uint256 _index) public view onlyOwner returns (address){
    require(_index <= getMinterLength() - 1, "JfToken: index out of bounds");
    return EnumerableSet.at(_minters, _index);
}

// modifier for mint function
modifier onlyMinter() {
    require(isMinter(msg.sender), "caller is not the minter");
    _;
}
}

```

#### ILfgSwapFactory.sol

```

pragma solidity >=0.5.0;

interface ILfgSwapFactory {
    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    function feeTo() external view returns (address);
    function feeToSetter() external view returns (address);

    function getPair(address tokenA, address tokenB) external view returns (address pair);
    function allPairs(uint) external view returns (address pair);
    function allPairsLength() external view returns (uint);

    function sortTokens(address tokenA, address tokenB) external pure returns (address token0, address token1);

    function pairFor(address tokenA, address tokenB) external view returns (address pair);

    function createPair(address tokenA, address tokenB) external returns (address pair);
}

```

```

function setFeeTo(address) external;
function setFeeToSetter(address) external;

}

```

## ILfgSwapRouter.sol

```

pragma solidity >=0.6.2;

interface ILfgSwapRouter {

    function factory() external pure returns (address);
    function WETH() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external payable returns (uint amountToken, uint amountETH, uint liquidity);
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountToken, uint amountETH);
    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityETHWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,

```



```

        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountToken, uint amountETH);
function swapExactTokensForTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external returns (uint[] memory amounts);
function swapTokensForExactTokens(
    uint amountOut,
    uint amountInMax,
    address[] calldata path,
    address to,
    uint deadline
) external returns (uint[] memory amounts);
function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadline
    external
    payable
    returns (uint[] memory amounts);
function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address
    external
    returns (uint[] memory amounts);
function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address
    external
    returns (uint[] memory amounts);
function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline
    external
    payable
    returns (uint[] memory amounts);

function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);
function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external pure returns (uint
function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external pure returns (uint
function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memo
function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[] memo

function removeLiquidityETHSupportingFeeOnTransferTokens(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline
) external returns (uint amountETH);
function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
) external returns (uint amountETH);

function swapExactTokensForTokensSupportingFeeOnTransferTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external;
function swapExactETHForTokensSupportingFeeOnTransferTokens(

```



```

        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external payable;
function swapExactTokensForETHSupportingFeeOnTransferTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external;
}

```

## IWETH.sol

```

pragma solidity >=0.5.0;

interface IWETH {
    function deposit() external payable;
    function transfer(address to, uint value) external returns (bool);
    function withdraw(uint) external;
    function balanceOf(address) external returns (uint256);
}

```

## ILfgSwapPair.sol

```

pragma solidity >=0.5.0;

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

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, by

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);
}

```

```

function MINIMUM_LIQUIDITY() external pure returns (uint);
function factory() external view returns (address);
function token0() external view returns (address);
function token1() external view returns (address);
function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTim
function price0CumulativeLast() external view returns (uint);
function price1CumulativeLast() external view returns (uint);
function kLast() external view returns (uint);

function mint(address to) external returns (uint liquidity);
function burn(address to) external returns (uint amount0, uint amount1);
function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
function skim(address to) external;
function sync() external;

function initialize(address, address) external;
}

```

## ISwapMining.sol

```

// SPDX-License-Identifier: MIT
pragma solidity =0.6.12;

interface ISwapMining {
    function swap(address _sender, address _tokenIn, address _tokenOut, uint256 _amountIn, uint256 _amo
}

```

## ILfgToken.sol

```

// SPDX-License-Identifier: MIT
pragma solidity =0.6.12;

import {IERC20 as SIERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";

interface ILfgToken is SIERC20 {
    function mint(address to, uint256 amount) external returns (bool);
}

```

## IERC20LfgSwap.sol

```

pragma solidity >=0.5.0;

interface IERC20LfgSwap {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
}

```

```

function nonces(address owner) external view returns (uint);

function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, by
}

```

## LfgSwapFactory.sol

```

pragma solidity =0.6.12;

import '../interface/ILfgSwapFactory.sol';
import './LfgSwapPair.sol';

contract LfgSwapFactory is ILfgSwapFactory {
    address public override feeTo;
    address public override feeToSetter;
    bytes32 public initCodeHash;

    mapping(address => mapping(address => address)) public override getPair;
    address[] public override allPairs;

    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    constructor(address _feeToSetter) public {
        feeToSetter = _feeToSetter;
        initCodeHash = keccak256(abi.encodePacked(type(LfgSwapPair).creationCode));
    }

    function allPairsLength() external override view returns (uint) {
        return allPairs.length;
    }

    function pairCodeHash() external pure returns (bytes32) {
        return keccak256(type(LfgSwapPair).creationCode);
    }

    // returns sorted token addresses, used to handle return values from pairs sorted in this order
    function sortTokens(address tokenA, address tokenB) public override pure returns (address token0,
        require(tokenA != tokenB, 'LfgSwapFactory: IDENTICAL_ADDRESSES');
        (token0, token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'LfgSwapFactory: ZERO_ADDRESS');
    }

    // calculates the CREATE2 address for a pair without making any external calls
    function pairFor(address tokenA, address tokenB) public override view returns (address pair) {
        (address token0, address token1) = sortTokens(tokenA, tokenB);
        pair = address(uint(keccak256(abi.encodePacked(
            hex'ff',
            address(this),
            keccak256(abi.encodePacked(token0, token1)),
            initCodeHash
        ))));
    }

    function createPair(address tokenA, address tokenB) external override returns (address pair) {
        require(tokenA != tokenB, 'LfgSwap: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'LfgSwap: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'LfgSwap: PAIR_EXISTS'); // single check is su
        bytes memory bytecode = type(LfgSwapPair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
    }
}

```

```

    LfgSwapPair(pair).initialize(token0, token1);
    getPair[token0][token1] = pair;
    getPair[token1][token0] = pair; // populate mapping in the reverse direction
    allPairs.push(pair);
    emit PairCreated(token0, token1, pair, allPairs.length);
}

function getSalt() public view returns(bytes32) {
    bytes memory bytecode = type(LfgSwapPair).creationCode;
    return keccak256(abi.encodePacked(bytecode));
}

function setFeeTo(address _feeTo) external override {
    require(msg.sender == feeToSetter, 'LfgSwap: FORBIDDEN');
    feeTo = _feeTo;
}

function setFeeToSetter(address _feeToSetter) external override {
    require(msg.sender == feeToSetter, 'LfgSwap: FORBIDDEN');
    feeToSetter = _feeToSetter;
}
}

```

## LfgSwapRouter.sol

```

pragma solidity =0.6.12;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";

import '../libraries/LfgSwapLibrary.sol';
import '../libraries/TransferHelper.sol';
import '../interface/ILfgSwapRouter.sol';
import '../interface/ILfgSwapFactory.sol';
import '../interface/IWETH.sol';
import '../interface/IERC20LfgSwap.sol';

contract LfgSwapRouter is ILfgSwapRouter, Ownable {
    using SafeMath for uint;

    address public override factory;
    address public override WETH;
    // address public immutable override WETH;

    modifier ensure(uint deadline) {
        require(deadline >= block.timestamp, 'LfgSwapRouter: EXPIRED');
        _;
    }

    constructor(address _factory, address _WETH) public {
        factory = _factory;
        WETH = _WETH;
    }

    receive() external payable {
        assert(msg.sender == WETH); // only accept ETH via fallback from the WETH contract
    }

    // **** ADD LIQUIDITY ****

```

```

function _addLiquidity(
    address tokenA,
    address tokenB,
    uint amountADesired,
    uint amountBDesired,
    uint amountAMin,
    uint amountBMin
) internal virtual returns (uint amountA, uint amountB) {
    // create the pair if it doesn't exist yet
    if (ILfgSwapFactory(factory).getPair(tokenA, tokenB) == address(0)) {
        ILfgSwapFactory(factory).createPair(tokenA, tokenB);
    }
    (uint reserveA, uint reserveB) = LfgSwapLibrary.getReserves(factory, tokenA, tokenB);
    if (reserveA == 0 && reserveB == 0) {
        (amountA, amountB) = (amountADesired, amountBDesired);
    } else {
        uint amountBOptimal = LfgSwapLibrary.quote(amountADesired, reserveA, reserveB);
        if (amountBOptimal <= amountBDesired) {
            require(amountBOptimal >= amountBMin, 'LfgSwapRouter: INSUFFICIENT_B_AMOUNT');
            (amountA, amountB) = (amountADesired, amountBOptimal);
        } else {
            uint amountAOptimal = LfgSwapLibrary.quote(amountBDesired, reserveB, reserveA);
            assert(amountAOptimal <= amountADesired);
            require(amountAOptimal >= amountAMin, 'LfgSwapRouter: INSUFFICIENT_A_AMOUNT');
            (amountA, amountB) = (amountAOptimal, amountBDesired);
        }
    }
}

function addLiquidity(
    address tokenA,
    address tokenB,
    uint amountADesired,
    uint amountBDesired,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline
) external virtual override ensure(deadline) returns (uint amountA, uint amountB, uint liquidity) {
    (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired, amountBDesired, amountAMin, amountBMin);
    address pair = LfgSwapLibrary.pairFor(factory, tokenA, tokenB);
    TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
    TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
    liquidity = ILfgSwapPair(pair).mint(to);
}

function addLiquidityETH(
    address token,
    uint amountTokenDesired,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline
) external virtual override payable ensure(deadline) returns (uint amountToken, uint amountETH, uint liquidity) {
    (amountToken, amountETH) = _addLiquidity(
        token,
        WETH,
        amountTokenDesired,
        msg.value,
        amountTokenMin,
        amountETHMin,
        to,
        deadline
    );
    address pair = LfgSwapLibrary.pairFor(factory, token, WETH);
    TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
    IWETH(WETH).deposit{value: amountETH}();
    assert(IWETH(WETH).transfer(pair, amountETH));
    liquidity = ILfgSwapPair(pair).mint(to);
    // refund dust eth, if any
}

```

```

    if (msg.value > amountETH) TransferHelper.safeTransferETH(msg.sender, msg.value - amountETH);
}

// **** REMOVE LIQUIDITY ****
function removeLiquidity(
    address tokenA,
    address tokenB,
    uint liquidity,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline
) public virtual override ensure(deadline) returns (uint amountA, uint amountB) {
    address pair = LfgSwapLibrary.pairFor(factory, tokenA, tokenB);
    ILfgSwapPair(pair).transferFrom(msg.sender, pair, liquidity); // send liquidity to pair
    (uint amount0, uint amount1) = ILfgSwapPair(pair).burn(to);
    (address token0,) = LfgSwapLibrary.sortTokens(tokenA, tokenB);
    (amountA, amountB) = tokenA == token0 ? (amount0, amount1) : (amount1, amount0);
    require(amountA >= amountAMin, 'LfgSwapRouter: INSUFFICIENT_A_AMOUNT');
    require(amountB >= amountBMin, 'LfgSwapRouter: INSUFFICIENT_B_AMOUNT');
}

function removeLiquidityETH(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline
) public virtual override ensure(deadline) returns (uint amountToken, uint amountETH) {
    (amountToken, amountETH) = removeLiquidity(
        token,
        WETH,
        liquidity,
        amountTokenMin,
        amountETHMin,
        address(this),
        deadline
    );
    TransferHelper.safeTransfer(token, to, amountToken);
    IWETH(WETH).withdraw(amountETH);
    TransferHelper.safeTransferETH(to, amountETH);
}

function removeLiquidityWithPermit(
    address tokenA,
    address tokenB,
    uint liquidity,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
) external virtual override returns (uint amountA, uint amountB) {
    address pair = LfgSwapLibrary.pairFor(factory, tokenA, tokenB);
    uint value = approveMax ? uint(-1) : liquidity;
    ILfgSwapPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
    (amountA, amountB) = removeLiquidity(tokenA, tokenB, liquidity, amountAMin, amountBMin, to, d
}

function removeLiquidityETHWithPermit(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
) external virtual override returns (uint amountToken, uint amountETH) {

```

```

        address pair = LfgSwapLibrary.pairFor(factory, token, WETH);
        uint value = approveMax ? uint(-1) : liquidity;
        ILfgSwapPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountToken, amountETH) = removeLiquidityETH(token, liquidity, amountTokenMin, amountETHMin,
    }

    // **** REMOVE LIQUIDITY (supporting fee-on-transfer tokens) ****
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountETH) {
        (, amountETH) = removeLiquidity(
            token,
            WETH,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, IERC20LfgSwap(token).balanceOf(address(this)));
        IWETH(WETH).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }
    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountETH) {
        address pair = LfgSwapLibrary.pairFor(factory, token, WETH);
        uint value = approveMax ? uint(-1) : liquidity;
        ILfgSwapPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        amountETH = removeLiquidityETHSupportingFeeOnTransferTokens(
            token, liquidity, amountTokenMin, amountETHMin, to, deadline
        );
    }

    // **** SWAP ****
    // requires the initial amount to have already been sent to the first pair
    function _swap(uint[] memory amounts, address[] memory path, address _to) internal virtual {
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = LfgSwapLibrary.sortTokens(input, output);
            uint amountOut = amounts[i + 1];
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOut) : (amountOut,
                address to = i < path.length - 2 ? LfgSwapLibrary.pairFor(factory, output, path[i + 2]) :

            ILfgSwapPair(LfgSwapLibrary.pairFor(factory, input, output)).swap(
                amount0Out, amount1Out, to, new bytes(0)
            );
        }
    }
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,

```



```

    uint deadline
) external virtual override ensure(deadline) returns (uint[] memory amounts) {
    amounts = LfgSwapLibrary.getAmountsOut(factory, amountIn, path);
    require(amounts[amounts.length - 1] >= amountOutMin, 'LfgSwapRouter: INSUFFICIENT_OUTPUT_AMOU

//swapExactTokensForTokens

TransferHelper.safeTransferFrom(
    path[0], msg.sender, LfgSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]
);
_swap(amounts, path, to);
}
function swapTokensForExactTokens(
    uint amountOut,
    uint amountInMax,
    address[] calldata path,
    address to,
    uint deadline
) external virtual override ensure(deadline) returns (uint[] memory amounts) {
    amounts = LfgSwapLibrary.getAmountsIn(factory, amountOut, path);
    require(amounts[0] <= amountInMax, 'LfgSwapRouter: EXCESSIVE_INPUT_AMOUNT');
    TransferHelper.safeTransferFrom(
        path[0], msg.sender, LfgSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]
    );
    _swap(amounts, path, to);
}
function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadl
external
virtual
override
payable
ensure(deadline)
returns (uint[] memory amounts)
{
    require(path[0] == WETH, 'LfgSwapRouter: INVALID_PATH');
    amounts = LfgSwapLibrary.getAmountsOut(factory, msg.value, path);
    require(amounts[amounts.length - 1] >= amountOutMin, 'LfgSwapRouter: INSUFFICIENT_OUTPUT_AMOU
    IWETH(WETH).deposit{value: amounts[0]}();
    assert(IWETH(WETH).transfer(LfgSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]));
    _swap(amounts, path, to);
}
function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address
external
virtual
override
ensure(deadline)
returns (uint[] memory amounts)
{
    require(path[path.length - 1] == WETH, 'LfgSwapRouter: INVALID_PATH');
    amounts = LfgSwapLibrary.getAmountsIn(factory, amountOut, path);
    require(amounts[0] <= amountInMax, 'LfgSwapRouter: EXCESSIVE_INPUT_AMOUNT');
    TransferHelper.safeTransferFrom(
        path[0], msg.sender, LfgSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]
    );
    _swap(amounts, path, address(this));
    IWETH(WETH).withdraw(amounts[amounts.length - 1]);
    TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
}
function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address
external
virtual
override
ensure(deadline)
returns (uint[] memory amounts)
{
    require(path[path.length - 1] == WETH, 'LfgSwapRouter: INVALID_PATH');

```

```

        amounts = LfgSwapLibrary.getAmountsOut(factory, amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'LfgSwapRouter: INSUFFICIENT_OUTPUT_AMOU
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, LfgSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        IWETH(WETH).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }
    function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline
        external
        virtual
        override
        payable
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[0] == WETH, 'LfgSwapRouter: INVALID_PATH');
        amounts = LfgSwapLibrary.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= msg.value, 'LfgSwapRouter: EXCESSIVE_INPUT_AMOUNT');
        IWETH(WETH).deposit{value: amounts[0]}();
        assert(IWETH(WETH).transfer(LfgSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]));
        _swap(amounts, path, to);
        // refund dust eth, if any
        if (msg.value > amounts[0]) TransferHelper.safeTransferETH(msg.sender, msg.value - amounts[0]
    }

    // **** SWAP (supporting fee-on-transfer tokens) ****
    // requires the initial amount to have already been sent to the first pair
    function _swapSupportingFeeOnTransferTokens(address[] memory path, address _to) internal virtual
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = LfgSwapLibrary.sortTokens(input, output);
            ILfgSwapPair pair = ILfgSwapPair(LfgSwapLibrary.pairFor(factory, input, output));
            uint amountInput;
            uint amountOutput;
            { // scope to avoid stack too deep errors
                (uint reserve0, uint reserve1,) = pair.getReserves();
                (uint reserveInput, uint reserveOutput) = input == token0 ? (reserve0, reserve1) : (reser
                amountInput = IERC20LfgSwap(input).balanceOf(address(pair)).sub(reserveInput);
                amountOutput = LfgSwapLibrary.getAmountOut(amountInput, reserveInput, reserveOutput);
            }

            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOutput) : (amount0
            address to = i < path.length - 2 ? LfgSwapLibrary.pairFor(factory, output, path[i + 2]) :
            pair.swap(amount0Out, amount1Out, to, new bytes(0));
        }
    }
    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) {
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, LfgSwapLibrary.pairFor(factory, path[0], path[1]), amountIn
        );
        uint balanceBefore = IERC20LfgSwap(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20LfgSwap(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >= amountOutMin,
            'LfgSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }
}
function swapExactETHForTokensSupportingFeeOnTransferTokens(

```

```

        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    )
    external
    virtual
    override
    payable
    ensure(deadline)
{
    require(path[0] == WETH, 'LfgSwapRouter: INVALID_PATH');
    uint amountIn = msg.value;
    IWETH(WETH).deposit{value: amountIn}();
    assert(IWETH(WETH).transfer(LfgSwapLibrary.pairFor(factory, path[0], path[1]), amountIn));
    uint balanceBefore = IERC20LfgSwap(path[path.length - 1]).balanceOf(to);
    _swapSupportingFeeOnTransferTokens(path, to);
    require(
        IERC20LfgSwap(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >= amountOutMin,
        'LfgSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT'
    );
}
function swapExactTokensForETHSupportingFeeOnTransferTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
)
    external
    virtual
    override
    ensure(deadline)
{
    require(path[path.length - 1] == WETH, 'LfgSwapRouter: INVALID_PATH');
    TransferHelper.safeTransferFrom(
        path[0], msg.sender, LfgSwapLibrary.pairFor(factory, path[0], path[1]), amountIn
    );
    _swapSupportingFeeOnTransferTokens(path, address(this));
    uint amountOut = IERC20LfgSwap(WETH).balanceOf(address(this));
    require(amountOut >= amountOutMin, 'LfgSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT');
    IWETH(WETH).withdraw(amountOut);
    TransferHelper.safeTransferETH(to, amountOut);
}

// **** LIBRARY FUNCTIONS ****
function quote(uint amountA, uint reserveA, uint reserveB) public pure virtual override returns (
    return LfgSwapLibrary.quote(amountA, reserveA, reserveB);
}

function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut)
    public
    pure
    virtual
    override
    returns (uint amountOut)
{
    return LfgSwapLibrary.getAmountOut(amountIn, reserveIn, reserveOut);
}

function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut)
    public
    pure
    virtual
    override
    returns (uint amountIn)

```

```

    {
        return LfgSwapLibrary.getAmountIn(amountOut, reserveIn, reserveOut);
    }

    function getAmountsOut(uint amountIn, address[] memory path)
        public
        view
        virtual
        override
        returns (uint[] memory amounts)
    {
        return LfgSwapLibrary.getAmountsOut(factory, amountIn, path);
    }

    function getAmountsIn(uint amountOut, address[] memory path)
        public
        view
        virtual
        override
        returns (uint[] memory amounts)
    {
        return LfgSwapLibrary.getAmountsIn(factory, amountOut, path);
    }
}

```

## LfgSwapPair.sol

```

pragma solidity =0.6.12;

import "../libraries/SafeMath.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "../libraries/UQ112x112.sol";
import "../interface/ILfgSwapFactory.sol";

interface IMigrator {
    // Return the desired amount of liquidity token that the migrator wants.
    function desiredLiquidity() external view returns (uint256);
}

interface ILfgSwapCallee {
    function jwapCall(address sender, uint amount0, uint amount1, bytes calldata data) external;
}

contract LfgSwapERC20 {
    using SafeMath for uint;

    string public constant name = 'LGF LP Token';
    string public constant symbol = 'LFG_LP';
    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 = 0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c6
    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 verifyingC
        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;
}

function transferFrom(address from, address to, uint value) external returns (bool) {
    if (allowance[from][msg.sender] != uint(-1)) {

        allowance[from][msg.sender] = allowance[from][msg.sender].sub(value);
    }
    _transfer(from, to, value);
    return true;
}

function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, by
    require(deadline >= block.timestamp, 'UniswapV2: EXPIRED');
    bytes32 digest = keccak256(
        abi.encodePacked(
            '\x19\x01',
            DOMAIN_SEPARATOR,
            keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value, nonces[owner]++, deadlin
        )
    )

```

```

    );
    address recoveredAddress = ecrecover(digest, v, r, s);
    require(recoveredAddress != address(0) && recoveredAddress == owner, 'UniswapV2: INVALID_SIGN
    _approve(owner, spender, value);
}
}

contract LfgSwapPair is LfgSwapERC20 {
    using SafeMath for uint;
    using UQ112x112 for uint224;

    uint public constant MINIMUM_LIQUIDITY = 10**3;
    bytes4 private constant SELECTOR = bytes4(keccak256(bytes('transfer(address,uint256)'))));

    address public factory;
    address public token0;
    address public token1;

    uint112 private reserve0;           // uses single storage slot, accessible via getReserves
    uint112 private reserve1;           // uses single storage slot, accessible via getReserves
    uint32 private blockTimestampLast; // uses single storage slot, accessible via getReserves

    uint public price0CumulativeLast;
    uint public price1CumulativeLast;
    uint public kLast; // reserve0 * reserve1, as of immediately after the most recent liquidity even

    uint private unlocked = 1;
    modifier lock() {
        require(unlocked == 1, 'LfgSwap: LOCKED');
        unlocked = 0;
        _;
        unlocked = 1;
    }

    function getReserves() public view returns (uint112 _reserve0, uint112 _reserve1, uint32 _blockTi
        _reserve0 = reserve0;
        _reserve1 = reserve1;
        _blockTimestampLast = blockTimestampLast;
    }

    function _safeTransfer(address token, address to, uint value) private {
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'LfgSwap: TRANSFER_FAILED'
    }

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    constructor() public {
        factory = msg.sender;
    }

    // called once by the factory at time of deployment
    function initialize(address _token0, address _token1) external {
        require(msg.sender == factory, 'LfgSwap: FORBIDDEN'); // sufficient check
        token0 = _token0;
        token1 = _token1;
    }

```

```

}

// update reserves and, on the first call per block, price accumulators
function _update(uint balance0, uint balance1, uint112 _reserve0, uint112 _reserve1) private {
    require(balance0 <= uint112(-1) && balance1 <= uint112(-1), 'LfgSwap: OVERFLOW');
    uint32 blockTimestamp = uint32(block.timestamp % 2**32);
    uint32 timeElapsed = blockTimestamp - blockTimestampLast; // overflow is desired
    if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0) {
        // * never overflows, and + overflow is desired
        price0CumulativeLast += uint(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) * timeElapsed;
        price1CumulativeLast += uint(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) * timeElapsed;
    }
    reserve0 = uint112(balance0);
    reserve1 = uint112(balance1);
    blockTimestampLast = blockTimestamp;
    emit Sync(reserve0, reserve1);
}

// if fee is on, mint liquidity equivalent to 1/6th of the growth in sqrt(k)
function _mintFee(uint112 _reserve0, uint112 _reserve1) private returns (bool feeOn) {
    address feeTo = ILfgSwapFactory(factory).feeTo();
    feeOn = feeTo != address(0);
    uint _kLast = kLast; // gas savings
    if (feeOn) {
        if (_kLast != 0) {
            uint rootK = SafeMath.sqrt(uint(_reserve0).mul(_reserve1));
            uint rootKLast = SafeMath.sqrt(_kLast);
            if (rootK > rootKLast) {
                uint numerator = totalSupply.mul(rootK.sub(rootKLast));
                uint denominator = rootK.mul(2).add(rootKLast);
                uint liquidity = numerator / denominator;
                if (liquidity > 0) _mint(feeTo, liquidity);
            }
        }
    } else if (_kLast != 0) {
        kLast = 0;
    }
}

// this low-level function should be called from a contract which performs important safety check
function mint(address to) external lock returns (uint liquidity) {
    (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
    uint balance0 = IERC20(token0).balanceOf(address(this));
    uint balance1 = IERC20(token1).balanceOf(address(this));
    uint amount0 = balance0.sub(_reserve0);
    uint amount1 = balance1.sub(_reserve1);

    bool feeOn = _mintFee(_reserve0, _reserve1);
    uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can u
    if (_totalSupply == 0) {
        address migrator = ILfgSwapFactory(factory).migrator();
        if (msg.sender == migrator) {
            liquidity = IMigrator(migrator).desiredLiquidity();
            require(liquidity > 0 && liquidity != uint256(-1), "Bad desired liquidity");
        } else {
            require(migrator == address(0), "Must not have migrator");
            liquidity = SafeMath.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
            _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first MINIMUM_LIQUIDITY
        }
    } else {
        liquidity = SafeMath.min(amount0.mul(_totalSupply) / _reserve0, amount1.mul(_totalSupply) / _reserve1);
    }
    require(liquidity > 0, 'LfgSwap: INSUFFICIENT_LIQUIDITY_MINTED');
    _mint(to, liquidity);

    _update(balance0, balance1, _reserve0, _reserve1);
}

```



```

    if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date

    emit Mint(msg.sender, amount0, amount1);
}
// this low-level function should be called from a contract which performs important safety check
function burn(address to) external lock returns (uint amount0, uint amount1) {
    (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
    address _token0 = token0; // gas savings
    address _token1 = token1; // gas savings
    uint balance0 = IERC20(_token0).balanceOf(address(this));
    uint balance1 = IERC20(_token1).balanceOf(address(this));
    uint liquidity = balanceOf[address(this)];

    bool feeOn = _mintFee(_reserve0, _reserve1);
    uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can u
    amount0 = liquidity.mul(balance0) / _totalSupply; // using balances ensures pro-rata distribu
    amount1 = liquidity.mul(balance1) / _totalSupply; // using balances ensures pro-rata distribu

    require(amount0 > 0 && amount1 > 0, 'LfgSwap: INSUFFICIENT_LIQUIDITY_BURNED');
    _burn(address(this), liquidity);
    _safeTransfer(_token0, to, amount0);
    _safeTransfer(_token1, to, amount1);
    balance0 = IERC20(_token0).balanceOf(address(this));
    balance1 = IERC20(_token1).balanceOf(address(this));

    _update(balance0, balance1, _reserve0, _reserve1);
    if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
    emit Burn(msg.sender, amount0, amount1, to);
}
// this low-level function should be called from a contract which performs important safety check
function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
    if( amount0Out > 0 || amount1Out > 0 ) {
        require(amount0Out > 0 || amount1Out > 0, 'LfgSwap: INSUFFICIENT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'LfgSwap: INSUFFICIENT_LIQUIDIT

        uint balance0;
        uint balance1;
        { // scope for _token{0,1}, avoids stack too deep errors
            address _token0 = token0;
            address _token1 = token1;
            require(to != _token0 && to != _token1, 'LfgSwap: INVALID_TO');
            if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer to
            if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer to
            if (data.length > 0) ILfgSwapCallee(to).jwapCall(msg.sender, amount0Out, amount1Out, data
            balance0 = IERC20(_token0).balanceOf(address(this));
            balance1 = IERC20(_token1).balanceOf(address(this));
        }
        uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out)
        uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out)
        require(amount0In > 0 || amount1In > 0, 'LfgSwap: INSUFFICIENT_INPUT_AMOUNT');
        { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
            uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
            uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
            require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000
        }

        _update(balance0, balance1, _reserve0, _reserve1);
        emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
    }
}

// force balances to match reserves
function skim(address to) external lock {
    address _token0 = token0; // gas savings

```

```

        address _token1 = token1; // gas savings
        _safeTransfer(_token0, to, IERC20(_token0).balanceOf(address(this)).sub(reserve0));
        _safeTransfer(_token1, to, IERC20(_token1).balanceOf(address(this)).sub(reserve1));
    }

    // force reserves to match balances
    function sync() external lock {
        _update(IERC20(token0).balanceOf(address(this)), IERC20(token1).balanceOf(address(this)), res
    }
}

```

## FixedPoint.sol

```

// // SPDX-License-Identifier: MIT
pragma solidity 0.6.12;
pragma experimental ABIEncoderV2;

library FixedPoint {
    // range: [0, 2**112 - 1]
    // resolution: 1 / 2**112
    struct uq112x112 {
        uint224 _x;
    }

    // range: [0, 2**144 - 1]
    // resolution: 1 / 2**112
    struct uq144x112 {
        uint _x;
    }

    uint8 private constant RESOLUTION = 112;

    // encode a uint112 as a UQ112x112
    function encode(uint112 x) internal pure returns (uq112x112 memory) {
        return uq112x112(uint224(x) << RESOLUTION);
    }

    // encodes a uint144 as a UQ144x112
    function encode144(uint144 x) internal pure returns (uq144x112 memory) {
        return uq144x112(uint256(x) << RESOLUTION);
    }

    // divide a UQ112x112 by a uint112, returning a UQ112x112
    function div(uq112x112 memory self, uint112 x) internal pure returns (uq112x112 memory) {
        require(x != 0, 'FixedPoint: DIV_BY_ZERO');
        return uq112x112(self._x / uint224(x));
    }

    // multiply a UQ112x112 by a uint, returning a UQ144x112
    // reverts on overflow
    function mul(uq112x112 memory self, uint y) internal pure returns (uq144x112 memory) {
        uint z;
        require(y == 0 || (z = uint(self._x) * y) / y == uint(self._x), "FixedPoint: MULTIPLICATION_O
        return uq144x112(z);
    }

    // returns a UQ112x112 which represents the ratio of the numerator to the denominator
    // equivalent to encode(numerator).div(denominator)
    function fraction(uint112 numerator, uint112 denominator) internal pure returns (uq112x112 memory) {
        require(denominator > 0, "FixedPoint: DIV_BY_ZERO");
        return uq112x112((uint224(numerator) << RESOLUTION) / denominator);
    }
}

```

```

// decode a UQ112x112 into a uint112 by truncating after the radix point
function decode(uq112x112 memory self) internal pure returns (uint112) {
    return uint112(self._x >> RESOLUTION);
}

// decode a UQ144x112 into a uint144 by truncating after the radix point
function decode144(uq144x112 memory self) internal pure returns (uint144) {
    return uint144(self._x >> RESOLUTION);
}
}

```

## UQ112x112.sol

```

pragma solidity =0.6.12;

// a library for handling binary fixed point numbers (https://en.wikipedia.org/wiki/Q_(number_format))

// range: [0, 2**112 - 1]
// resolution: 1 / 2**112

library UQ112x112 {
    uint224 constant Q112 = 2**112;

    // encode a uint112 as a UQ112x112
    function encode(uint112 y) internal pure returns (uint224 z) {
        z = uint224(y) * Q112; // never overflows
    }

    // divide a UQ112x112 by a uint112, returning a UQ112x112
    function uqdiv(uint224 x, uint112 y) internal pure returns (uint224 z) {
        z = x / uint224(y);
    }
}

```

## LfgSwapLibrary.sol

```

pragma solidity >=0.5.0;

import '../interface/ILfgSwapPair.sol';
import '../interface/ILfgSwapFactory.sol';

import "@openzeppelin/contracts/math/SafeMath.sol";

library LfgSwapLibrary {
    using SafeMath for uint;

    // returns sorted token addresses, used to handle return values from pairs sorted in this order
    function sortTokens(address tokenA, address tokenB) internal pure returns (address token0, address token1) {
        require(tokenA != tokenB, 'UniswapV2Library: IDENTICAL_ADDRESSES');
        (token0, token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'UniswapV2Library: ZERO_ADDRESS');
    }

    // calculates the CREATE2 address for a pair without making any external calls
    function pairFor(address factory, address tokenA, address tokenB) internal view returns (address pair) {
        (address token0, address token1) = sortTokens(tokenA, tokenB);
        pair = ILfgSwapFactory(factory).getPair(token0, token1);

        // = address(uint(keccak256(abi.encodePacked(
        //     hex'ff',
        //     factory,

```

```

//      keccak256(abi.encodePacked(token0, token1)),
//      hex'e18a34eb0e04b04f7a0ac29a6e80748dca96319b42c54d679cb821dca90c6303' // init code
//      ))));
}

// fetches and sorts the reserves for a pair
function getReserves(address factory, address tokenA, address tokenB) internal view returns (uint
    (address token0,) = sortTokens(tokenA, tokenB);
    (uint reserve0, uint reserve1,) = ILfgSwapPair(pairFor(factory, tokenA, tokenB)).getReserves(
    (reserveA, reserveB) = tokenA == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
}

// given some amount of an asset and pair reserves, returns an equivalent amount of the other ass
function quote(uint amountA, uint reserveA, uint reserveB) internal pure returns (uint amountB) {
    require(amountA > 0, 'UniswapV2Library: INSUFFICIENT_AMOUNT');
    require(reserveA > 0 && reserveB > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
    amountB = amountA.mul(reserveB) / reserveA;
}

// given an input amount of an asset and pair reserves, returns the maximum output amount of the
function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) internal pure returns (uint
    require(amountIn > 0, 'UniswapV2Library: INSUFFICIENT_INPUT_AMOUNT');
    require(reserveIn > 0 && reserveOut > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
    uint amountInWithFee = amountIn.mul(997);
    uint numerator = amountInWithFee.mul(reserveOut);
    uint denominator = reserveIn.mul(1000).add(amountInWithFee);
    amountOut = numerator / denominator;
}

// given an output amount of an asset and pair reserves, returns a required input amount of the o
function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) internal pure returns (uint
    require(amountOut > 0, 'UniswapV2Library: INSUFFICIENT_OUTPUT_AMOUNT');
    require(reserveIn > 0 && reserveOut > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
    uint numerator = reserveIn.mul(amountOut).mul(1000);
    uint denominator = reserveOut.sub(amountOut).mul(997);
    amountIn = (numerator / denominator).add(1);
}

// performs chained getAmountOut calculations on any number of pairs
function getAmountsOut(address factory, uint amountIn, address[] memory path) internal view retur
    require(path.length >= 2, 'UniswapV2Library: INVALID_PATH');
    amounts = new uint[](path.length);
    amounts[0] = amountIn;
    for (uint i; i < path.length - 1; i++) {
        (uint reserveIn, uint reserveOut) = getReserves(factory, path[i], path[i + 1]);
        amounts[i + 1] = getAmountOut(amounts[i], reserveIn, reserveOut);
    }
}

// performs chained getAmountIn calculations on any number of pairs
function getAmountsIn(address factory, uint amountOut, address[] memory path) internal view retur
    require(path.length >= 2, 'UniswapV2Library: INVALID_PATH');
    amounts = new uint[](path.length);
    amounts[amounts.length - 1] = amountOut;
    for (uint i = path.length - 1; i > 0; i--) {
        (uint reserveIn, uint reserveOut) = getReserves(factory, path[i - 1], path[i]);
        amounts[i - 1] = getAmountIn(amounts[i], reserveIn, reserveOut);
    }
}
}

```

```

pragma solidity >=0.5.0 <0.8.0;

library SafeMath {
    uint256 constant WAD = 10 ** 18;
    uint256 constant RAY = 10 ** 27;

    function wad() public pure returns (uint256) {
        return WAD;
    }

    function ray() public pure returns (uint256) {
        return RAY;
    }

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
    }
}

```

```

    return a % b;
}

function min(uint256 a, uint256 b) internal pure returns (uint256) {
    return a <= b ? a : b;
}

function max(uint256 a, uint256 b) internal pure returns (uint256) {
    return a >= b ? a : b;
}

function sqrt(uint256 a) internal pure returns (uint256 b) {
    if (a > 3) {
        b = a;
        uint256 x = a / 2 + 1;
        while (x < b) {
            b = x;
            x = (a / x + x) / 2;
        }
    } else if (a != 0) {
        b = 1;
    }
}

function wmul(uint256 a, uint256 b) internal pure returns (uint256) {
    return mul(a, b) / WAD;
}

function wmulRound(uint256 a, uint256 b) internal pure returns (uint256) {
    return add(mul(a, b), WAD / 2) / WAD;
}

function rmul(uint256 a, uint256 b) internal pure returns (uint256) {
    return mul(a, b) / RAY;
}

function rmulRound(uint256 a, uint256 b) internal pure returns (uint256) {
    return add(mul(a, b), RAY / 2) / RAY;
}

function wdiv(uint256 a, uint256 b) internal pure returns (uint256) {
    return div(mul(a, WAD), b);
}

function wdivRound(uint256 a, uint256 b) internal pure returns (uint256) {
    return add(mul(a, WAD), b / 2) / b;
}

function rdiv(uint256 a, uint256 b) internal pure returns (uint256) {
    return div(mul(a, RAY), b);
}

function rdivRound(uint256 a, uint256 b) internal pure returns (uint256) {
    return add(mul(a, RAY), b / 2) / b;
}

function wpow(uint256 x, uint256 n) internal pure returns (uint256) {
    uint256 result = WAD;
    while (n > 0) {
        if (n % 2 != 0) {
            result = wmul(result, x);
        }
        x = wmul(x, x);
        n /= 2;
    }
    return result;
}

```

```

    }

    function rpow(uint256 x, uint256 n) internal pure returns (uint256) {
        uint256 result = RAY;
        while (n > 0) {
            if (n % 2 != 0) {
                result = rmul(result, x);
            }
            x = rmul(x, x);
            n /= 2;
        }
        return result;
    }
}

```

## TransferHelper.sol

```

// SPDX-License-Identifier: GPL-3.0-or-later

pragma solidity >=0.6.0;

// helper methods for interacting with ERC20 tokens and sending ETH that do not consistently return t
library TransferHelper {
    function safeApprove(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: APPROVE_F
    }

    function safeTransfer(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: TRANSFER_
    }

    function safeTransferFrom(address token, address from, address to, uint value) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to, v
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: TRANSFER_
    }

    function safeTransferETH(address to, uint value) internal {
        (bool success,) = to.call{value:value}(new bytes(0));
        require(success, 'TransferHelper: ETH_TRANSFER_FAILED');
    }
}

```

## Analysis of audit results

### Re-Entrancy

- **Description:**

One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the



contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Arithmetic Over/Under Flows

- **Description:**

The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Unexpected Blockchain Currency

- **Description:**

Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.

- **Detection results:**

PASSED!

- **Security suggestion:** no.

## Delegatecall

- **Description:**

The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.

- **Detection results:**

PASSED!

- **Security suggestion:** no.

## Default Visibilities

---

- **Description:**

Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whether a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devastating vulnerabilities in smart contracts as will be discussed in this section.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Entropy Illusion

---

- **Description:**

All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no `rand()` function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## External Contract Referencing

---

- **Description:**

One of the benefits of the global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Unsolved TODO comments

---

- **Description:**  
Check for Unsolved TODO comments
- **Detection results:**

PASSED!

- **Security suggestion:**  
no.

## Short Address/Parameter Attack

---

- **Description:**  
This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

PASSED!

- **Security suggestion:**  
no.

## Unchecked CALL Return Values

---

- **Description:**  
There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

PASSED!

- **Security suggestion:**  
no.

## Race Conditions / Front Running

---

- **Description:**  
The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.
- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Denial Of Service (DOS)

---

- **Description:**

This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Block Timestamp Manipulation

---

- **Description:**

Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Constructors with Care

---

- **Description:**

Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Unintialised Storage Pointers

---

- **Description:**

The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default

types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately initialising variables.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Floating Points and Numerical Precision

---

- **Description:**

As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## tx.origin Authentication

---

- **Description:**

Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Permission restrictions

---

- **Description:**

Contract managers who can control liquidity or pledge pools, etc., or impose unreasonable restrictions on other users.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

[armors.io](https://armors.io)

[contact@armors.io](mailto:contact@armors.io)

