知道创宇

密级 ★ | 项目组公开

智能合约审计报告

安全状态

安全

★ ★ ★ ★ ★

主测人： 知道创宇区块链安全研究团队

版本说明

修订内容	时间	修订者	版本号
编写文档	20201010	知道创宇区块链安全研究团队	V1.0

文档信息

文档名称	文档版本	文档编号	保密级别
COCK 智能合约审计报告	V1.0	COCK-ZNHY-20201010	项目组公开

声明

创宇仅就本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后发生或存在的事实，创宇无法判断其智能合约安全状况，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于信息提供者截至本报告出具时向创宇提供的文件和资料。创宇假设:已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，创宇对由此而导致的损失和不利影响不承担任何责任。

目录

1. 综述 - 6 -

2. 代码漏洞分析 - 8 -

2.1 漏洞等级分布 - 8 -

2.2 审计结果汇总说明 - 9 -

3. 业务安全性检测 - 11 -

3.1 波场 USDT 转账函数 Transfer 返回值差异【通过】 - 11 -

3.2 事件通知数据不一致【低危】 - 11 -

4. 代码基本漏洞检测 - 13 -

4.1 重入攻击检测【通过】 - 13 -

4.2 重放攻击检测【通过】 - 13 -

4.3 重排攻击检测【通过】 - 13 -

4.4 数值溢出检测【通过】 - 14 -

4.5 算术精度误差【通过】 - 14 -

4.6 访问控制检测【通过】 - 15 -

4.7 tx.origin 身份验证【通过】 - 15 -

4.8 call 注入攻击【通过】 - 15 -

4.9 返回值调用验证【通过】 - 16 -

4.10 未初始化的储存指针【通过】 - 16 -

4.11 错误使用随机数【通过】 - 17 -

4.12 交易顺序依赖【通过】 - 17 -

4.13 拒绝服务攻击【通过】 - 17 -
4.14 假充值漏洞【通过】 - 18 -
4.15 增发代币漏洞【通过】 - 18 -
4.16 冻结账户绕过【通过】 - 18 -
4.17 编译器版本安全【通过】 - 19 -
4.18 不推荐的编码方式【通过】 - 19 -
4.19 冗余代码【通过】 - 19 -
4.20 安全算数库的使用【通过】 - 19 -
4.21 require/assert 的使用【通过】 - 20 -
4.22 energy 消耗检测【通过】 - 20 -
4.23 fallback 函数安全【通过】 - 20 -
4.24 owner 权限控制【通过】 - 20 -
4.25 低级函数安全【通过】 - 21 -
4.26 变量覆盖【通过】 - 21 -
4.27 时间戳依赖攻击【通过】 - 21 -
4.28 不安全的接口使用【通过】 - 22 -
5. 附录 A：合约代码 - 22 -
6. 附录 B：安全风险评级标准 - 77 -
7. 附录 C：智能合约安全审计工具简介 - 78 -
7.1 Manticore - 78 -
7.2 Oyente - 78 -
7.3 securify.sh - 78 -

7.4 Echidna ..... - 78 -

7.5 MAIAN ..... - 78 -

7.6 ethersplay ..... - 79 -

7.7 ida-evm ..... - 79 -

7.8 Remix-ide ..... - 79 -

7.9 知道创宇区块链安全审计人员专用工具包 ..... - 79 -

1. 综述

本次报告有效测试时间是从 2020 年 10 月 9 日开始到 2020 年 10 月 10 日结束，在此期间针对 **COCK 智能合约代码**的安全性和规范性进行审计并以此作为报告统计依据。

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第三、四章节）进行了全面的分析，存在事件通知数据不一致的问题，由于该问题不属于安全问题，故综合评定为**通过**。

本次智能合约安全审计结果： 通过

由于本次测试过程在非生产环境下进行，所有代码均为最新备份，测试过程均与相关接口人进行沟通，并在操作风险可控的情况下进行相关测试操作，以规避测试过程中的生产运营风险、代码安全风险。

本次测试的目标信息：

条目	描述
Token 名称	COCK
代码类型	代币代码、DeFi 协议代码、波场智能合约
代码语言	Solidity

合约文件及哈希：

合约文件	MD5
CockLpPoolReward.sol	9075e8bd2123dda37b33b5c6e28c5e4d
CockTail.sol	f0f4a1812e685c6a3e444920b7e20ff6
JfiLpPoolReward.sol	0127474ee9aa456b9a360c5fed9e3392
JfiPool.sol	d3dd3ce4b0062bb4c6b96954787142ea
SunLpPoolReward.sol	e2f94674612c5a49f5488771e20627ba

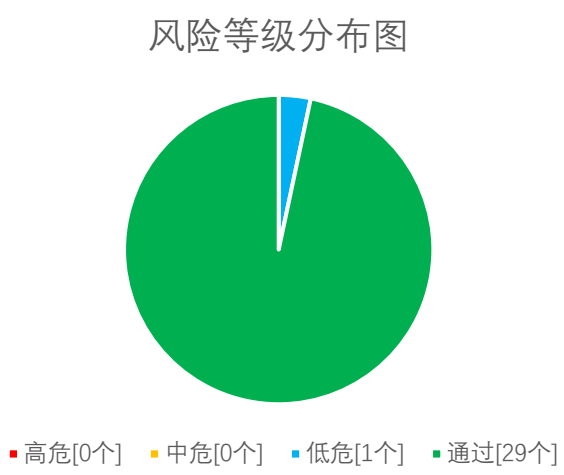
TrxPool.sol	675fe24814994e1d596a2f084349de03
UsdjPool.sol	037c6ecbe00cced05acfdf49373c8353

2. 代码漏洞分析

2.1 漏洞等级分布

本次漏洞风险按等级统计：

安全风险等级个数统计表			
高危	中危	低危	通过
0	0	1	29



2.2 审计结果汇总说明

审计结果			
审计项目	审计内容	状态	描述
智能合约	波场 USDT 转账函数 Transfer 返回值差异	通过	经检测，不存在该安全问题。
	事件通知数据不一致	低危（通过）	经检测，存在该安全问题。但该问题不属于安全问题，故通过。
	重入攻击检测	通过	经检测，不存在该安全问题。
	重放攻击检测	通过	经检测，不存在该安全问题。
	数值溢出检测	通过	经检测，不存在该安全问题。
	算数精度误差	通过	经检测，不存在该安全问题。
	访问控制缺陷检测	通过	经检测，不存在该安全问题。
	tx.progin 身份验证	通过	经检测，不存在该安全问题。
	call 注入攻击	通过	经检测，不存在该安全问题。
	返回值调用验证	通过	经检测，不存在该安全问题。
	未初始化的存储指针	通过	经检测，不存在该安全问题。
	错误使用随机数检测	通过	经检测，不存在该安全问题。
	交易顺序依赖检测	通过	经检测，不存在该安全问题。
	拒绝服务攻击检测	通过	经检测，不存在该安全问题。
	业务安全性检测	通过	经检测，不存在该安全问题。
	假充值漏洞检测	通过	经检测，不存在该安全问题。
	增发代币漏洞检测	通过	经检测，不存在该安全问题。
	冻结账户绕过检测	通过	经检测，不存在该安全问题。
	编译器版本安全	通过	经检测，不存在该安全问题。
	不推荐的编码方式	通过	经检测，不存在该安全问题。

	冗余代码	通过	经检测，不存在该安全问题。
	安全算数库的使用	通过	经检测，不存在该安全问题。
	require/assert 的使用	通过	经检测，不存在该安全问题。
	energy 消耗检测	通过	经检测，不存在该安全问题。
	fallback 函数安全	通过	经检测，不存在该安全问题。
	owner 权限控制	通过	经检测，不存在该安全问题。
	低级函数安全	通过	经检测，不存在该安全问题。
	变量覆盖	通过	经检测，不存在该安全问题。
	时间戳依赖攻击	通过	经检测，不存在该安全问题。
	不安全的接口使用	通过	经检测，不存在该安全问题。

3. 业务安全性检测

3.1 波场 USDT 转账函数 Transfer 返回值差异【通过】

在波场 USDT 代币合约项目中，transfer 函数未使用 TIP20 规范的写法导致函数在执行时未返回对应的值，最终返回默认的 false，从而导致在使用 safeTransfer 调用 USDT 的 transfer 时永远都只返回 false，导致用户无法提现。详情请参考我们团队的技术分析文章：https://paper.seebug.org/1337/

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

3.2 事件通知数据不一致【低危】

智能合约的通知事件需要反映真实的情况，多个同类型合约中，事件通知的数据需一致，避免产生误解。

检测结果：经检测，智能合约代码中存在该安全问题。JfiLpPoolReward.sol 文件的第 675 行的事件通知的奖励数据与其他合约不一致。

```
function getReward() public updateReward(msg.sender)    checkStart{
    uint256 reward = earned(msg.sender);
    if (reward > 0) {
        rewards[msg.sender] = 0;
        cock.safeTransfer(msg.sender, reward.mul(90).div(100));
        emit RewardPaid(msg.sender, reward.mul(90).div(100));
        //knownsec//  与其他合约的事件通知数据不一致,其他合约均为 reward  总量而不是 90%
        //knownsec//  建议将 reward.mul(90).div(100)改为 reward
        //社区奖励
```

```
            cock.safeTransfer(addrCBO,reward.mul(10).div(100));
        }
    }
```

安全建议：统一该事件通知的奖励数据。

4. 代码基本漏洞检测

4.1 重入攻击检测【通过】

重入漏洞是最著名的智能合约漏洞。

在 Solidity 中，调用其他地址的函数或者给合约地址转账，都会把自己的地址作为被调合约的 msg.sender 传递过去。此时，如果传递的 energy 足够的话，就可能会被对方进行重入攻击，在波场中对合约地址调用 transfer/sender，只会传递 2300 的 Energy，这不足以发起一次重入攻击。所以要一定避免通过 call.value 的方式，调用未知的合约地址。因为 call.value 可以传入远大于 2300 的 energy。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.2 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击

在资产管理体系中，常有委托管理的情况，委托人将资产给受托人管理，委托人支付一定的费用给受托人。这个业务场景在智能合约中也比较普遍。。

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.3 重排攻击检测【通过】

重排攻击是指矿工或其他方试图通过将自己的信息插入列表(list)或映射

(mapping)中来与智能合约参与者进行“竞争”，从而使攻击者有机会将自己的信息存储到合约中。

检测结果:经检测，智能合约代码中不存在相关漏洞。

安全建议:无。

4.4 数值溢出检测【通过】

智能合约中的算数问题是指整数溢出和整数下溢。

Solidity 最多能处理 256 位的数字（2^256-1），最大数字增加 1 会溢出得到 0。同样，当数字为无符号类型时，0 减去 1 会下溢得到最大数字值。

整数溢出和下溢不是一种新类型的漏洞，但它们在智能合约中尤其危险。溢出情况会导致不正确的结果，特别是如果可能性未被预期，可能会影响程序的可靠性和安全性。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.5 算术精度误差【通过】

Solidity 作为一门编程语言具备和普通编程语言相似的数据结构设计，比如：变量、常量、函数、数组、函数、结构体等等，Solidity 和普通编程语言也有一个较大的区别——Solidity 没有浮点型，且 Solidity 所有的数值运算结果都只会是整数，不会出现小数的情况，同时也不允许定义小数类型数据。合约中的数值运算必不可少，而数值运算的设计有可能造成相对误差，例如同级运算：5/2*10=20，而 5*10/2=25，从而产生误差，在数据更大时产生的误差也会更大，更明显。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.6 访问控制检测【通过】

合约中不同函数应设置合理的权限

检查合约中各函数是否正确使用了 public、private 等关键词进行可见性修饰，检查合约是否正确定义并使用了 modifier 对关键函数进行访问限制，避免越权导致的问题。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.7 tx.origin 身份验证【通过】

tx.origin 是 Solidity 的一个全局变量，它遍历整个调用栈并返回最初发送调用（或事务）的帐户的地址。在智能合约中使用此变量进行身份验证会使合约容易受到类似网络钓鱼的攻击。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.8 call 注入攻击【通过】

call 函数调用时，应该做严格的权限控制，或直接写死 call 调用的函数。

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.9 返回值调用验证【通过】

此问题多出现在和转币相关的智能合约中，故又称作静默失败发送或未经检查发送。

在 Solidity 中存在 transfer()、send()、call.value()等转币方法，都可以用于向某一地址发送 trx，其区别在于： transfer 发送失败时会 throw，并且进行状态回滚；只会传递 2300energy 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300energy 供调用，防止重入攻击；call.value 发送失败时会返回 false；传递所有可用 energy 进行调用（可通过传入 feeLimit 参数进行限制），不能有效防止重入攻击。

如果在代码中没有检查以上 send 和 call.value 转币函数的返回值，合约会继续执行后面的代码，可能由于 trx 发送失败而导致意外的结果。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.10 未初始化的储存指针【通过】

在 solidity 中允许一个特殊的数据结构为 struct 结构体，而函数内的局部变量默认使用 storage 或 memory 储存。

而存在 storage(存储器)和 memory(内存)是两个不同的概念，solidity 允许指针指向一个未初始化的引用，而未初始化的局部 stroage 会导致变量指向其他储存变量，导致变量覆盖，甚至其他更严重的后果，在开发中应该避免在函数中初始化 struct 变量。

检测结果：经检测，智能合约代码不使用结构体，不存在该问题。

安全建议：无。

4.11 错误使用随机数【通过】

智能合约中可能需要使用随机数，虽然 Solidity 提供的函数和变量可以访问明显难以预测的值，如 block.number 和 block.timestamp，但是它们通常或者比看起来更公开，或者受到矿工的影响，即这些随机数在一定程度上是可预测的，所以恶意用户通常可以复制它并依靠其不可预知性来攻击该功能。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.12 交易顺序依赖【通过】

由于矿工总是通过代表外部拥有地址（EOA）的代码获取 energy 费用，因此用户可以指定更高的费用以便更快地开展交易。由于波场区块链是公开的，每个人都可以看到其他人未决交易的内容。这意味着，如果某个用户提交了一个有价值的解决方案，恶意用户可以窃取该解决方案并以较高的费用复制其交易，以抢占原始解决方案。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无

4.13 拒绝服务攻击【通过】

在波场的世界中，拒绝服务是致命的，遭受该类型攻击的智能合约可能永远

无法恢复正常工作状态。导致智能合约拒绝服务的原因可能有很多种，包括在作为交易接收方时的恶意行为，人为增加计算功能所需 energy 导致 energy 耗尽，滥用访问控制访问智能合约的 private 组件，利用混淆和疏忽等等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.14 假充值漏洞【通过】

在代币合约的 transfer 函数对转账发起人(msg.sender)的余额检查用的是 if 判断方式，当 balances[msg.sender] < value 时进入 else 逻辑部分并 return false，最终没有抛出异常，我们认为仅 if/else 这种温和的判断方式在 transfer 这类敏感函数场景中是一种不严谨的编码方式。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.15 增发代币漏洞【通过】

检查在初始化代币总量后，代币合约中是否存在可能使代币总量增加的函数。

检测结果：经检测，智能合约代码中存在增发代币的功能，但由于流动性挖矿需要增发代币，故通过。

安全建议：无。

4.16 冻结账户绕过【通过】

检查代币合约中在转移代币时，是否存在未校验代币来源账户、发起账户、

目标账户是否被冻结的操作。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.17 编译器版本安全【通过】

检查合约代码实现中是否使用了安全的编译器版本

检测结果：经检测，智能合约代码中制定了编译器版本 0.5.0 以上，不存在该安全问题。

安全建议：无。

4.18 不推荐的编码方式【通过】

检查合约代码实现中是否有官方不推荐或弃用的编码方式

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.19 冗余代码【通过】

检查合约代码实现中是否包含冗余代码

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.20 安全算数库的使用【通过】

检查合约代码实现中是否使用了 SafeMath 安全算数库

检测结果：经检测，智能合约代码中已使用 SafeMath 安全算数库，不存在该安全问题。

安全建议：无。

4.21 require/assert 的使用【通过】

检查合约代码实现中 require 和 assert 语句使用的合理性

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.22 energy 消耗检测【通过】

检查 energy 的消耗是否超过区块最大限制

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.23 fallback 函数安全【通过】

检查合约代码实现中是否正确使用 fallback 函数

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.24 owner 权限控制【通过】

检查合约代码实现中的 owner 是否具有过高的权限。例如，任意修改其他账户余额等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.25 低级函数安全【通过】

检查合约代码实现中低级函数（call/delegatecall）的使用是否存在安全漏洞

call 函数的执行上下文是在被调用的合约中；而 delegatecall 函数的执行上下文是在当前调用该函数的合约中

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.26 变量覆盖【通过】

检查合约代码实现中是否存在变量覆盖导致的安全问题

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.27 时间戳依赖攻击【通过】

数据块的时间戳通常来说都是使用矿工的本地时间，而这个时间大约能有 900 秒的范围波动，当其他节点接受一个新区块时，只需要验证时间戳是否晚于之前的区块并且与本地时间误差在 900 秒以内。一个矿工可以通过设置区块的时间戳来尽可能满足有利于他的条件来从中获利。

检查合约代码实现中是否存在有依赖于时间戳的关键功能

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.28 不安全的接口使用【通过】

检查合约代码实现中是否使用了不安全的接口

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

5. 附录 A：合约代码

本次测试代码来源：

CockLpPoolReward.sol

```
pragma solidity ^0.5.0;//knownsec//  指定编译器版本
/**
 * @dev Standard math utilities missing in the Solidity language.
```

```
 */
library Math {
    /**
     * @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, so we distribute
        return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
    }
}

// File: @openzeppelin/contracts/math/SafeMath.sol

pragma solidity ^0.5.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0._
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;
```

```
        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    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;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    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;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
```

```
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

// File: @openzeppelin/contracts/GSN/Context.sol

pragma solidity ^0.5.0;

/*
 * @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 GSN 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.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

// File: @openzeppelin/contracts/ownership/Ownable.sol

pragma solidity ^0.5.0;

/**
 * @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.
 *
 * 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.
 */
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 () internal {
        _owner = _msgSender();
        emit OwnershipTransferred(address(0), _owner);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
```

```
        return _msgSender() == _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 onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = 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 onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol

pragma solidity ^0.5.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
interface IERC20 {
    /**
     * @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 `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);
    function mint(address account, uint amount) external;

    /**
     * @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 `sender` to `recipient` 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 sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @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);
}

// File: @openzeppelin/contracts/utils/Address.sol

pragma solidity ^0.5.5;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * This test is non-exhaustive, and there may be false-negatives: during the
     * execution of a contract's constructor, its address will be reported as
     * not containing a contract.
     *
     * IMPORTANT: It is unsafe to assume that an address for which this
     * function returns false is an externally-owned account (EOA) and not a
     * contract.
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }

    /**
     * @dev Converts an `address` into `address payable`. Note that this is
     * simply a type cast: the actual underlying value is not changed.
     *
     * _Available since v2.4.0._
     */
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
     *
```

```
     * _Available since v2.4.0._
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}

// File: @openzeppelin/contracts/token/ERC20/SafeERC20.sol

pragma solidity ^0.5.0;



/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for ERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased
allowance below zero");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing the requirement
     * on the return value: the return value is optional (but if data is returned, it must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism, since
        // we're implementing it ourselves.

        // A Solidity high level call has three parts:
        //   1. The target address is checked to verify it contains contract code
        //   2. The call itself is made, and success asserted
        //   3. The return value is decoded, which in turn checks the size of the returned data.
        // solhint-disable-next-line max-line-length
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
```

```
}
// File: contracts/IRewardDistributionRecipient.sol

pragma solidity ^0.5.0;


contract IRewardDistributionRecipient is Ownable {
    address rewardDistribution;

    function notifyRewardAmount(uint256 reward) external;

    modifier onlyRewardDistribution() {
        require(_msgSender() == rewardDistribution, "Caller is not reward distribution");
        _;
    }

    function setRewardDistribution(address _rewardDistribution)
        external
        onlyOwner
    {
        rewardDistribution = _rewardDistribution;
    }
}

// File: contracts/CurveRewards.sol

pragma solidity ^0.5.0;


contract LPTokenWrapper {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

     //LPtoken 合约账户 dragon/trx
    //IERC20 public y = IERC20(0x419BDFD02372A4FF014CB0DC9821F60D1FC38C3C9C);
     //cock/trx
          IERC20                                    public                                  y                                   =
IERC20(0x412F51DF67B7C9FF5CB62D0BF801CC2E0EC4C6DDC6);//knownsec//
TEHQqgsjLZgFVJuXiF6srBC7yv7ewApeJo  未验证合约


    uint256 private _totalSupply;
    mapping(address => uint256) private _balances;

    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    function stake(uint256 amount) public {
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);
        y.safeTransferFrom(msg.sender, address(this), amount);
    }

    function withdraw(uint256 amount) public {
         require(_balances[msg.sender].sub(amount)>=0,"withdraw amount error")
        _totalSupply = _totalSupply.sub(amount);
        _balances[msg.sender] = _balances[msg.sender].sub(amount);
        y.transfer(msg.sender, amount);
    }
}

contract CockLpPoolReward is LPTokenWrapper, IRewardDistributionRecipient {
     // IERC20 public dragon = IERC20(0x41F554962E889A8BD93D5E9FF9FD85123566569F09);
      //主网 cock 代币 TMYigtLSE5uaqWLRQQAQzHuWdXAutFpfN8
     IERC20 public cock = IERC20(0x417EFFE43C5197F34A0073943B25D55D9DF92190B5);
    uint256 public constant DURATION = 4 weeks;

    uint256 public initreward = 3500*1e18;
    uint256 public starttime = 1600331096;//1599829200


    uint256 public periodFinish = 0;
    uint256 public rewardRate = 0;
    uint256 public lastUpdateTime;
    uint256 public rewardPerTokenStored;
    mapping(address => uint256) public userRewardPerTokenPaid;
    mapping(address => uint256) public rewards;


     //社区建设地址
```

```
    address public addrCBO;
    //是否已经初始化
    uint8 public initialized = 0;

    event RewardAdded(uint256 reward);
    event Staked(address indexed user, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);
    event RewardPaid(address indexed user, uint256 reward);

    modifier updateReward(address account) {
        rewardPerTokenStored = rewardPerToken();
        lastUpdateTime = lastTimeRewardApplicable();
        if (account != address(0)) {
            rewards[account] = earned(account);
            userRewardPerTokenPaid[account] = rewardPerTokenStored;
        }
        _;
    }

    function lastTimeRewardApplicable() public view returns (uint256) {
        return Math.min(block.timestamp, periodFinish);
    }

    function rewardPerToken() public view returns (uint256) {
        if (totalSupply() == 0) {
            return rewardPerTokenStored;
        }
        return
            rewardPerTokenStored.add(
                lastTimeRewardApplicable()
                    .sub(lastUpdateTime)
                    .mul(rewardRate)
                    .mul(1e18)
                    .div(totalSupply())
            );
    }

    function earned(address account) public view returns (uint256) {
        return
            balanceOf(account)
                .mul(rewardPerToken().sub(userRewardPerTokenPaid[account]))
                .div(1e18)
                .add(rewards[account]);
    }

    // stake visibility is public as overriding LPTokenWrapper's stake() function
    function stake(uint256 amount) public updateReward(msg.sender) checkStart checkEnd{
        require(amount > 0, "Cannot stake 0");
        super.stake(amount);
        emit Staked(msg.sender, amount);
    }

    function withdraw(uint256 amount) public updateReward(msg.sender)    checkStart{
        require(amount > 0, "Cannot withdraw 0");
        super.withdraw(amount);
        emit Withdrawn(msg.sender, amount);
    }

    function exit() external {
        withdraw(balanceOf(msg.sender));
        getReward();
    }

    function getReward() public updateReward(msg.sender)    checkStart{
        uint256 reward = earned(msg.sender);
        if (reward > 0) {
            rewards[msg.sender] = 0;
            cock.safeTransfer(msg.sender, reward.mul(90).div(100));
            emit RewardPaid(msg.sender, reward);
              //社区奖励
              cock.safeTransfer(addrCBO,reward.mul(10).div(100));//knownsec// 10%的社区奖励
        }
    }

    //modifier checkhalve(){
      //    if (block.timestamp >= periodFinish) {
            // initreward = initreward.mul(50).div(100);
        //        cock.mint(address(this),initreward);

        //        rewardRate = initreward.div(DURATION);
        //        periodFinish = block.timestamp.add(DURATION);
        //        emit RewardAdded(initreward);
      //    }
      //    _;
    // }

    //开采结束算法
```

```
    modifier checkEnd(){
        require(periodFinish >= starttime,"ended");
        _;
    }

     function initContract(address community) public onlyOwner{//knownsec//  合约初始化,仅 owner 调用
        require(initialized == 0, " contract has initialized");
        addrCBO = community;
        cock.mint(address(this),initreward);//knownsec//  挖出初始奖励额
        rewardRate = initreward.div(DURATION);
        periodFinish = starttime.add(DURATION);
        initialized = 1;
        emit RewardAdded(initreward);
     }


    modifier checkStart(){
        require(block.timestamp > starttime,"not start");
        //require(initialized == 1,"contract not init");
        _;
    }

    function notifyRewardAmount(uint256 reward)
        external
        onlyRewardDistribution
        updateReward(address(0))
    {
        if (block.timestamp >= periodFinish) {
            rewardRate = reward.div(DURATION);
        } else {
            uint256 remaining = periodFinish.sub(block.timestamp);
            uint256 leftover = remaining.mul(rewardRate);
            rewardRate = reward.add(leftover).div(DURATION);
        }
        cock.mint(address(this),reward);
        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp.add(DURATION);
        emit RewardAdded(reward);
    }

        //更改社区发展奖励地址
     function dev(address community) public onlyOwner{//knownsec//  仅 owner 调用
        //require(msg.sender == addrCBO, "dev: wut?");
        addrCBO = community;
    }
}
```

CockTail.sol

```
pragma solidity ^0.5.4;

interface IERC20 {
    function totalSupply() external view returns (uint);
    function balanceOf(address account) external view returns (uint);
    function transfer(address recipient, uint amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint);
    function approve(address spender, uint amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint value);
    event Approval(address indexed owner, address indexed spender, uint value);
}

contract Context {
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }
}

contract ERC20 is Context, IERC20 {
    using SafeMath for uint;

    mapping (address => uint) private _balances;

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

    uint private _totalSupply;
    function totalSupply() public view returns (uint) {
        return _totalSupply;
    }
    function balanceOf(address account) public view returns (uint) {
        return _balances[account];
    }
    function transfer(address recipient, uint amount) public returns (bool) {
        _transfer(_msgSender(), recipient, amount);
```

```
        return true;
    }
    function allowance(address owner, address spender) public view returns (uint) {
        return _allowances[owner][spender];
    }
    function approve(address spender, uint amount) public returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }
    function transferFrom(address sender, address recipient, uint amount) public returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));
        return true;
    }
    function increaseAllowance(address spender, uint addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }
    function decreaseAllowance(address spender, uint subtractedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero"));
        return true;
    }
    function _transfer(address sender, address recipient, uint amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }
    function _mint(address account, uint amount) internal {
        require(account != address(0), "ERC20: mint to the zero address");

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
    function _burn(address account, uint amount) internal {
        require(account != address(0), "ERC20: burn from the zero address");

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }
    function _approve(address owner, address spender, uint amount) internal {
        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);
    }
}

contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }
    function name() public view returns (string memory) {
        return _name;
    }
    function symbol() public view returns (string memory) {
        return _symbol;
    }
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}

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

        return c;
    }
    function sub(uint a, uint b) internal pure returns (uint) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
    function sub(uint a, uint b, string memory errorMessage) internal pure returns (uint) {
```

```
            require(b <= a, errorMessage);
            uint c = a - b;

            return c;
        }
        function mul(uint a, uint b) internal pure returns (uint) {
            if (a == 0) {
                return 0;
            }

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

            return c;
        }
        function div(uint a, uint b) internal pure returns (uint) {
            return div(a, b, "SafeMath: division by zero");
        }
        function div(uint a, uint b, string memory errorMessage) internal pure returns (uint) {
            // Solidity only automatically asserts when dividing by 0
            require(b > 0, errorMessage);
            uint c = a / b;

            return c;
        }
}

library Address {


        function isContract(address account) internal view returns (bool) {


            //bytes32 codehash;
            //bytes32                                                    accountHash                                                    =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
            // solhint-disable-next-line no-inline-assembly
            //assembly { codehash := extcodehash(account) }
            //return (codehash != 0x0 && codehash != accountHash);
              uint size;

            assembly { size := extcodesize(account) }

            return size > 0;
        }
}

library SafeERC20 {
        using SafeMath for uint;
        using Address for address;

        function safeTransfer(IERC20 token, address to, uint value) internal {
            callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
        }

        function safeTransferFrom(IERC20 token, address from, address to, uint value) internal {
            callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
        }

        function safeApprove(IERC20 token, address spender, uint value) internal {
            require((value == 0) || (token.allowance(address(this), spender) == 0),
                "SafeERC20: approve from non-zero to non-zero allowance"
            );
            callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
        }
        function callOptionalReturn(IERC20 token, bytes memory data) private {
            require(address(token).isContract(), "SafeERC20: call to non-contract");

            // solhint-disable-next-line avoid-low-level-calls
            (bool success, bytes memory returndata) = address(token).call(data);
            require(success, "SafeERC20: low-level call failed");

            if (returndata.length > 0) { // Return data is optional
                // solhint-disable-next-line max-line-length
                require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
            }
        }
}

contract CockTail is ERC20, ERC20Detailed {
    using SafeERC20 for IERC20;
    using Address for address;
    using SafeMath for uint;


    address public governance;
```

```
    mapping (address => bool) public minters;

    constructor () public ERC20Detailed("COCKTAIL", "COCK", 18) {
        governance = msg.sender;
    }

    function mint(address account, uint amount) public {
        require(minters[msg.sender], "!minter");
        _mint(account, amount);
    }

    function setGovernance(address _governance) public {
        require(msg.sender == governance, "!governance");
        governance = _governance;
    }

    function addMinter(address _minter) public {
        require(msg.sender == governance, "!governance");
        minters[_minter] = true;
    }

    function removeMinter(address _minter) public {
        require(msg.sender == governance, "!governance");
        minters[_minter] = false;
    }
}
```

JfiLpPoolReward.sol

```
pragma solidity ^0.5.0;

/**
 * @dev Standard math utilities missing in the Solidity language.
 */
library Math {
    /**
     * @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, so we distribute
        return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
    }
}

// File: @openzeppelin/contracts/math/SafeMath.sol

pragma solidity ^0.5.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
```

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

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0._
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    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;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
```

```
     * _Available since v2.4.0._
     */
    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;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

// File: @openzeppelin/contracts/GSN/Context.sol

pragma solidity ^0.5.0;

/*
 * @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 GSN 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.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

// File: @openzeppelin/contracts/ownership/Ownable.sol

pragma solidity ^0.5.0;

/**
 * @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.
 *
 * 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.
 */
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 () internal {
        _owner = _msgSender();
        emit OwnershipTransferred(address(0), _owner);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return _msgSender() == _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 onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = 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 onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol

pragma solidity ^0.5.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
interface IERC20 {
    /**
     * @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 `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);
    function mint(address account, uint amount) external;

    /**
     * @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 `sender` to `recipient` 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 sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @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);
}

// File: @openzeppelin/contracts/utils/Address.sol

pragma solidity ^0.5.5;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * This test is non-exhaustive, and there may be false-negatives: during the
     * execution of a contract's constructor, its address will be reported as
     * not containing a contract.
     *
     * IMPORTANT: It is unsafe to assume that an address for which this
     * function returns false is an externally-owned account (EOA) and not a
     * contract.
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
```

```
            bytes32 codehash;
            bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
            // solhint-disable-next-line no-inline-assembly
            assembly { codehash := extcodehash(account) }
            return (codehash != 0x0 && codehash != accountHash);
        }

        /**
         * @dev Converts an `address` into `address payable`. Note that this is
         * simply a type cast: the actual underlying value is not changed.
         *
         * _Available since v2.4.0._
         */
        function toPayable(address account) internal pure returns (address payable) {
            return address(uint160(account));
        }

        /**
         * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
         * `recipient`, forwarding all available gas and reverting on errors.
         *
         * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
         * of certain opcodes, possibly making contracts go over the 2300 gas limit
         * imposed by `transfer`, making them unable to receive funds via
         * `transfer`. {sendValue} removes this limitation.
         *
         * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
         *
         * IMPORTANT: because control is transferred to `recipient`, care must be
         * taken to not create reentrancy vulnerabilities. Consider using
         * {ReentrancyGuard} or the
         * `https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-
pattern[checks-effects-interactions pattern].
         *
         * _Available since v2.4.0._
         */
        function sendValue(address payable recipient, uint256 amount) internal {
            require(address(this).balance >= amount, "Address: insufficient balance");

            // solhint-disable-next-line avoid-call-value
            (bool success, ) = recipient.call.value(amount)("");
            require(success, "Address: unable to send value, recipient may have reverted");
        }
}

// File: @openzeppelin/contracts/token/ERC20/SafeERC20.sol

pragma solidity ^0.5.0;



/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for ERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
```

```
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased allowance below zero");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing the requirement
     * on the return value: the return value is optional (but if data is returned, it must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism, since
        // we're implementing it ourselves.

        // A Solidity high level call has three parts:
        //    1. The target address is checked to verify it contains contract code
        //    2. The call itself is made, and success asserted
        //    3. The return value is decoded, which in turn checks the size of the returned data.
        // solhint-disable-next-line max-line-length
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

// File: contracts/IRewardDistributionRecipient.sol

pragma solidity ^0.5.0;


contract IRewardDistributionRecipient is Ownable {
    address rewardDistribution;

    function notifyRewardAmount(uint256 reward) external;

    modifier onlyRewardDistribution() {
        require(_msgSender() == rewardDistribution, "Caller is not reward distribution");
        _;
    }

    function setRewardDistribution(address _rewardDistribution)
        external
        onlyOwner
    {
        rewardDistribution = _rewardDistribution;
    }
}

// File: contracts/CurveRewards.sol

pragma solidity ^0.5.0;


contract LPTokenWrapper {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;
     //jfi_trx lp token
    IERC20 public y = IERC20(0x41015AB982849DBA5B043C46AAB14D1359E5B8A746);//knownsec// JFI-TRX

    uint256 private _totalSupply;
    mapping(address => uint256) private _balances;

    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    function stake(uint256 amount) public {
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);
        y.safeTransferFrom(msg.sender, address(this), amount);
```

```
    }

    function withdraw(uint256 amount) public {
            require(_balances[msg.sender].sub(amount)>=0,"withdraw amount error");
            _totalSupply = _totalSupply.sub(amount);
            _balances[msg.sender] = _balances[msg.sender].sub(amount);
            y.transfer(msg.sender, amount);
    }
}

contract JfiLpPoolReward is LPTokenWrapper, IRewardDistributionRecipient {


     //cock 代币
    IERC20 public cock = IERC20(0x417EFFE43C5197F34A0073943B25D55D9DF92190B5);
    uint256 public constant DURATION = 4 weeks;

    uint256 public initreward = 750*1e18;
    uint256 public starttime = 1600331096;//1599908400
    uint256 public periodFinish = 0;
    uint256 public rewardRate = 0;
    uint256 public lastUpdateTime;
    uint256 public rewardPerTokenStored;
    mapping(address => uint256) public userRewardPerTokenPaid;
    mapping(address => uint256) public rewards;



     //社区建设地址
     address public addrCBO;
     //是否已经初始化
     uint8 public initialized = 0;


    event RewardAdded(uint256 reward);
    event Staked(address indexed user, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);
    event RewardPaid(address indexed user, uint256 reward);

    modifier updateReward(address account) {
        rewardPerTokenStored = rewardPerToken();
        lastUpdateTime = lastTimeRewardApplicable();
        if (account != address(0)) {
            rewards[account] = earned(account);
            userRewardPerTokenPaid[account] = rewardPerTokenStored;
        }
        _;
    }

    function lastTimeRewardApplicable() public view returns (uint256) {
        return Math.min(block.timestamp, periodFinish);
    }

    function rewardPerToken() public view returns (uint256) {
        if (totalSupply() == 0) {
            return rewardPerTokenStored;
        }
        return
            rewardPerTokenStored.add(
                lastTimeRewardApplicable()
                    .sub(lastUpdateTime)
                    .mul(rewardRate)
                    .mul(1e18)
                    .div(totalSupply())
            );
    }

    function earned(address account) public view returns (uint256) {
        return
            balanceOf(account)
                .mul(rewardPerToken().sub(userRewardPerTokenPaid[account]))
                .div(1e18)
                .add(rewards[account]);
    }

    // stake visibility is public as overriding LPTokenWrapper's stake() function
    function stake(uint256 amount) public updateReward(msg.sender)    checkStart{
        require(amount > 0, "Cannot stake 0");
        super.stake(amount);
        emit Staked(msg.sender, amount);
    }

    function withdraw(uint256 amount) public updateReward(msg.sender)    checkStart checkEnd{
        require(amount > 0, "Cannot withdraw 0");
        super.withdraw(amount);
        emit Withdrawn(msg.sender, amount);
    }
```

```
    function exit() external {
        withdraw(balanceOf(msg.sender));
        getReward();
    }

    function getReward() public updateReward(msg.sender)    checkStart{
        uint256 reward = earned(msg.sender);
        if (reward > 0) {
            rewards[msg.sender] = 0;
            cock.safeTransfer(msg.sender, reward.mul(90).div(100));
            emit RewardPaid(msg.sender, reward.mul(90).div(100));
            //knownsec//  与其他合约的事件通知数据不一致,其他合约均为reward 总量而不是 90%
            //knownsec//  建议将 reward.mul(90).div(100)改为 reward
              //社区奖励
              cock.safeTransfer(addrCBO,reward.mul(10).div(100));
        }
    }

   // modifier checkhalve(){
    //      if (block.timestamp >= periodFinish) {
    //          initreward = initreward.mul(50).div(100);
    //          cock.mint(address(this),initreward);

    //          rewardRate = initreward.div(DURATION);
    //          periodFinish = block.timestamp.add(DURATION);
    //          emit RewardAdded(initreward);
    //      }
    //      _;
  //  }


        //开采结束算法
    modifier checkEnd(){
        require(periodFinish >= starttime,"ended");
        _;
    }

     function initContract(address community) public onlyOwner{//knownsec//  合约初始化,仅 owner 调用
        require(initialized == 0, " contract has initialized");
        addrCBO = community;
        cock.mint(address(this),initreward);
        rewardRate = initreward.div(DURATION);
        periodFinish = starttime.add(DURATION);
        initialized = 1;
        emit RewardAdded(initreward);
     }

    modifier checkStart(){
        require(block.timestamp > starttime,"not start");
        _;
    }

    function notifyRewardAmount(uint256 reward)
        external
        onlyRewardDistribution
        updateReward(address(0))
    {
        if (block.timestamp >= periodFinish) {
            rewardRate = reward.div(DURATION);
        } else {
            uint256 remaining = periodFinish.sub(block.timestamp);
            uint256 leftover = remaining.mul(rewardRate);
            rewardRate = reward.add(leftover).div(DURATION);
        }
        cock.mint(address(this),reward);
        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp.add(DURATION);
        emit RewardAdded(reward);
    }
}
```

JfiPool.sol

```
pragma solidity ^0.5.0;

/**
 * @dev Standard math utilities missing in the Solidity language.
 */
library Math {
    /**
     * @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, so we distribute
        return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
    }
}

// File: @openzeppelin/contracts/math/SafeMath.sol

pragma solidity ^0.5.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0._
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
```

```
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    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;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    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;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
```

```
        }
}

// File: @openzeppelin/contracts/GSN/Context.sol

pragma solidity ^0.5.0;

/*
 * @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 GSN 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.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

// File: @openzeppelin/contracts/ownership/Ownable.sol

pragma solidity ^0.5.0;

/**
 * @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.
 *
 * 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.
 */
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 () internal {
        _owner = _msgSender();
        emit OwnershipTransferred(address(0), _owner);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return _msgSender() == _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 onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = 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 onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol

pragma solidity ^0.5.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
interface IERC20 {
    /**
     * @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 `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);
    function mint(address account, uint amount) external;

    /**
     * @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 `sender` to `recipient` 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 sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @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);
}

// File: @openzeppelin/contracts/utils/Address.sol

pragma solidity ^0.5.5;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * This test is non-exhaustive, and there may be false-negatives: during the
     * execution of a contract's constructor, its address will be reported as
     * not containing a contract.
     *
     * IMPORTANT: It is unsafe to assume that an address for which this
     * function returns false is an externally-owned account (EOA) and not a
     * contract.
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        // bytes32 codehash;
        //bytes32                                    accountHash                                    =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        // assembly { codehash := extcodehash(account) }
        //return (codehash != 0x0 && codehash != accountHash);


        uint size;

        assembly { size := extcodesize(account) }

        return size > 0;
    }

    /**
     * @dev Converts an `address` into `address payable`. Note that this is
     * simply a type cast: the actual underlying value is not changed.
     *
     * _Available since v2.4.0._
     */
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-
pattern[checks-effects-interactions pattern].
     *
```

```
     * _Available since v2.4.0._
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}

// File: @openzeppelin/contracts/token/ERC20/SafeERC20.sol

pragma solidity ^0.5.0;



/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for ERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased
allowance below zero");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing the requirement
     * on the return value: the return value is optional (but if data is returned, it must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism, since
        // we're implementing it ourselves.

        // A Solidity high level call has three parts:
        //   1. The target address is checked to verify it contains contract code
        //   2. The call itself is made, and success asserted
        //   3. The return value is decoded, which in turn checks the size of the returned data.
        // solhint-disable-next-line max-line-length
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
```

```
}
// File: contracts/IRewardDistributionRecipient.sol

pragma solidity ^0.5.0;


contract IRewardDistributionRecipient is Ownable {
    address rewardDistribution;
    function notifyRewardAmount(uint256 reward) external;

    modifier onlyRewardDistribution() {
        require(_msgSender() == rewardDistribution, "Caller is not reward distribution");
        _;
    }
    function setRewardDistribution(address _rewardDistribution)
        external
        onlyOwner
    {
        rewardDistribution = _rewardDistribution;
    }
}

// File: contracts/CurveRewards.sol

pragma solidity ^0.5.0;


contract LPTokenWrapper {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    IERC20 public y = IERC20(0x854aBd86aB4b76EC440EB7F4EEeba79E6D499417); //JFI:


    uint256 private _totalSupply;

    mapping(address => uint256) private _balances;


    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    function stake(uint256 amount) public {
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);
        y.safeTransferFrom(msg.sender, address(this), amount);
    }
    function withdraw(uint256 amount) public {
        require(_balances[msg.sender].sub(amount)>=0,'withdraw amount error');
        _totalSupply = _totalSupply.sub(amount);
        _balances[msg.sender] = _balances[msg.sender].sub(amount);
        y.transfer(msg.sender,amount);
    }
}

contract JFIPoolReward is LPTokenWrapper, IRewardDistributionRecipient {


    IERC20 public cock = IERC20(0x412D6284C1EB4EBE9B1A14304560740 23A082D06C5);
    uint256 public constant DURATION = 4 weeks;
    uint8 public initialized = 0;
    uint256 public initreward = 1500*1e18;
    uint256 public starttime = 1600696800;
    uint256 public periodFinish = 0;
    uint256 public rewardRate = 0;
    uint256 public lastUpdateTime;
    uint256 public rewardPerTokenStored;
    mapping(address => uint256) public userRewardPerTokenPaid;
    mapping(address => uint256) public rewards;


    address public addrCBO;

    event RewardAdded(uint256 reward);
    event Staked(address indexed user, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);
    event RewardPaid(address indexed user, uint256 reward);
```

```
    modifier updateReward(address account) {

        rewardPerTokenStored = rewardPerToken();

        lastUpdateTime = lastTimeRewardApplicable();
        if (account != address(0)) {
            rewards[account] = earned(account);
            userRewardPerTokenPaid[account] = rewardPerTokenStored;
        }
        _;
    }

    function lastTimeRewardApplicable() public view returns (uint256) {
        return Math.min(block.timestamp, periodFinish);
    }


    function rewardPerToken() public view returns (uint256) {
          if (totalSupply() == 0) {
            return rewardPerTokenStored;
        }
          return
            rewardPerTokenStored.add(
                lastTimeRewardApplicable()
                    .sub(lastUpdateTime)
                    .mul(rewardRate)
                    .mul(1e18)
                    .div(totalSupply())
            );
    }

    function earned(address account) public view returns (uint256) {
        return
            balanceOf(account)
                .mul(rewardPerToken().sub(userRewardPerTokenPaid[account]))
                .div(1e18)
                .add(rewards[account]);
    }

    // stake visibility is public as overriding LPTokenWrapper's stake() function

    function stake(uint256 amount) public updateReward(msg.sender)    checkStart checkEnd{
        require(amount > 0, "Cannot stake 0");
        super.stake(amount);
        emit Staked(msg.sender, amount);
    }

    function withdraw(uint256 amount) public updateReward(msg.sender)    checkStart{
        require(amount > 0, "Cannot withdraw 0");
          super.withdraw(amount);
        emit Withdrawn(msg.sender, amount);
    }

    function exit() external {
        withdraw(balanceOf(msg.sender));
        getReward();
    }

    function getReward() public updateReward(msg.sender)    checkStart{
        uint256 reward = earned(msg.sender);
        if (reward > 0) {
            rewards[msg.sender] = 0;
            cock.safeTransfer(msg.sender, reward.mul(90).div(100));
              cock.safeTransfer(addrCBO,reward.mul(10).div(100));
            emit RewardPaid(msg.sender, reward);
        }
    }

    modifier checkEnd(){
          require(periodFinish >= starttime,"ended");
        _;
    }

     function initContract(address community) public onlyOwner{//knownsec//  合约初始化,仅 owner 调用
          require(initialized == 0, " contract has initialized");
          addrCBO = community;
          cock.mint(address(this),initreward);
          rewardRate = initreward.div(DURATION);
          periodFinish = starttime.add(DURATION);
          initialized = 1;
          emit RewardAdded(initreward);
     }

    modifier checkStart(){
        require(block.timestamp > starttime,"not start");
          require(initialized == 1,"contract not init");
        _;
```

```
    }

    function notifyRewardAmount(uint256 reward)
        external
        onlyRewardDistribution
        updateReward(address(0))
          onlyOwner
    {
        if (block.timestamp >= periodFinish) {
            rewardRate = reward.div(DURATION);
        } else {
            uint256 remaining = periodFinish.sub(block.timestamp);
            uint256 leftover = remaining.mul(rewardRate);
            rewardRate = reward.add(leftover).div(DURATION);
        }
        cock.mint(address(this),reward);
        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp.add(DURATION);
        emit RewardAdded(reward);
    }

     function dev(address community) public onlyOwner{
        //require(msg.sender == addrCBO, "dev: wut?");
        addrCBO = community;
    }
}
```

SunLpPoolReward.sol

```
pragma solidity ^0.5.0;

/**
  * @dev Standard math utilities missing in the Solidity language.
  */
library Math {
    /**
      * @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, so we distribute
        return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
    }
}

// File: @openzeppelin/contracts/math/SafeMath.sol

pragma solidity ^0.5.0;

/**
  * @dev Wrappers over Solidity's arithmetic operations with added overflow
  * checks.
  *
  * Arithmetic operations in Solidity wrap on overflow. This can easily result
  * in bugs, because programmers usually assume that an overflow raises an
  * error, which is the standard behavior in high level programming languages.
  * `SafeMath` restores this intuition by reverting the transaction when 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.
  */
library SafeMath {
    /**
      * @dev Returns the addition of two unsigned integers, reverting on
      * overflow.
      *
      * Counterpart to Solidity's `+` operator.
      *
      * Requirements:
      * - Addition cannot overflow.
      */
```

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

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0._
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    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;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
```

```
     * _Available since v2.4.0._
     */
    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;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

// File: @openzeppelin/contracts/GSN/Context.sol

pragma solidity ^0.5.0;

/*
 * @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 GSN 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.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

// File: @openzeppelin/contracts/ownership/Ownable.sol

pragma solidity ^0.5.0;

/**
 * @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.
 *
 * 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.
 */
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 () internal {
        _owner = _msgSender();
        emit OwnershipTransferred(address(0), _owner);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return _msgSender() == _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 onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = 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 onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol

pragma solidity ^0.5.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
interface IERC20 {
    /**
     * @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 `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);
    function mint(address account, uint amount) external;

    /**
     * @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 `sender` to `recipient` 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 sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @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);
}

// File: @openzeppelin/contracts/utils/Address.sol

pragma solidity ^0.5.5;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * This test is non-exhaustive, and there may be false-negatives: during the
     * execution of a contract's constructor, its address will be reported as
     * not containing a contract.
     *
     * IMPORTANT: It is unsafe to assume that an address for which this
     * function returns false is an externally-owned account (EOA) and not a
     * contract.
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
```

```
            bytes32 codehash;
            bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
            // solhint-disable-next-line no-inline-assembly
            assembly { codehash := extcodehash(account) }
            return (codehash != 0x0 && codehash != accountHash);
        }

        /**
         * @dev Converts an `address` into `address payable`. Note that this is
         * simply a type cast: the actual underlying value is not changed.
         *
         * _Available since v2.4.0._
         */
        function toPayable(address account) internal pure returns (address payable) {
            return address(uint160(account));
        }

        /**
         * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
         * `recipient`, forwarding all available gas and reverting on errors.
         *
         * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
         * of certain opcodes, possibly making contracts go over the 2300 gas limit
         * imposed by `transfer`, making them unable to receive funds via
         * `transfer`. {sendValue} removes this limitation.
         *
         * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
         *
         * IMPORTANT: because control is transferred to `recipient`, care must be
         * taken to not create reentrancy vulnerabilities. Consider using
         * {ReentrancyGuard} or the
         * `https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-
pattern[checks-effects-interactions pattern].
         *
         * _Available since v2.4.0._
         */
        function sendValue(address payable recipient, uint256 amount) internal {
            require(address(this).balance >= amount, "Address: insufficient balance");

            // solhint-disable-next-line avoid-call-value
            (bool success, ) = recipient.call.value(amount)("");
            require(success, "Address: unable to send value, recipient may have reverted");
        }
}

// File: @openzeppelin/contracts/token/ERC20/SafeERC20.sol

pragma solidity ^0.5.0;



/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for ERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
```

```
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased allowance below zero");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing the requirement
     * on the return value: the return value is optional (but if data is returned, it must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism, since
        // we're implementing it ourselves.

        // A Solidity high level call has three parts:
        //    1. The target address is checked to verify it contains contract code
        //    2. The call itself is made, and success asserted
        //    3. The return value is decoded, which in turn checks the size of the returned data.
        // solhint-disable-next-line max-line-length
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

// File: contracts/IRewardDistributionRecipient.sol

pragma solidity ^0.5.0;


contract IRewardDistributionRecipient is Ownable {
    address rewardDistribution;

    function notifyRewardAmount(uint256 reward) external;

    modifier onlyRewardDistribution() {
        require(_msgSender() == rewardDistribution, "Caller is not reward distribution");
        _;
    }

    function setRewardDistribution(address _rewardDistribution)
        external
        onlyOwner
    {
        rewardDistribution = _rewardDistribution;
    }
}

// File: contracts/CurveRewards.sol

pragma solidity ^0.5.0;


contract LPTokenWrapper {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;
     //jfi_trx lp token
    IERC20 public y = IERC20(0x41015AB982849DBA5B043C46AAB14D1359E5B8A746);

    uint256 private _totalSupply;
    mapping(address => uint256) private _balances;

    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    function stake(uint256 amount) public {
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);
        y.safeTransferFrom(msg.sender, address(this), amount);
    }
```

```
    function withdraw(uint256 amount) public {
            require(_balances[msg.sender].sub(amount)>=0,"withdraw amount error");
        _totalSupply = _totalSupply.sub(amount);
        _balances[msg.sender] = _balances[msg.sender].sub(amount);
        y.transfer(msg.sender, amount);
    }
}

contract JfiLpPoolReward is LPTokenWrapper, IRewardDistributionRecipient {


     //cock 代币
    IERC20 public cock = IERC20(0x417EFFE43C5197F34A0073943B25D55D9DF92190B5);
    uint256 public constant DURATION = 4 weeks;

    uint256 public initreward = 750*1e18;
    uint256 public starttime = 1600331096;//1599908400
    uint256 public periodFinish = 0;
    uint256 public rewardRate = 0;
    uint256 public lastUpdateTime;
    uint256 public rewardPerTokenStored;
    mapping(address => uint256) public userRewardPerTokenPaid;
    mapping(address => uint256) public rewards;



     //社区建设地址
     address public addrCBO;
     //是否已经初始化
     uint8 public initialized = 0;


    event RewardAdded(uint256 reward);
    event Staked(address indexed user, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);
    event RewardPaid(address indexed user, uint256 reward);

    modifier updateReward(address account) {
        rewardPerTokenStored = rewardPerToken();
        lastUpdateTime = lastTimeRewardApplicable();
        if (account != address(0)) {
            rewards[account] = earned(account);
            userRewardPerTokenPaid[account] = rewardPerTokenStored;
        }
        _;
    }

    function lastTimeRewardApplicable() public view returns (uint256) {
        return Math.min(block.timestamp, periodFinish);
    }

    function rewardPerToken() public view returns (uint256) {
        if (totalSupply() == 0) {
            return rewardPerTokenStored;
        }
        return
            rewardPerTokenStored.add(
                lastTimeRewardApplicable()
                    .sub(lastUpdateTime)
                    .mul(rewardRate)
                    .mul(1e18)
                    .div(totalSupply())
            );
    }

    function earned(address account) public view returns (uint256) {
        return
            balanceOf(account)
                .mul(rewardPerToken().sub(userRewardPerTokenPaid[account]))
                .div(1e18)
                .add(rewards[account]);
    }

    // stake visibility is public as overriding LPTokenWrapper's stake() function
    function stake(uint256 amount) public updateReward(msg.sender)    checkStart{
        require(amount > 0, "Cannot stake 0");
        super.stake(amount);
        emit Staked(msg.sender, amount);
    }

    function withdraw(uint256 amount) public updateReward(msg.sender)    checkStart checkEnd{
        require(amount > 0, "Cannot withdraw 0");
        super.withdraw(amount);
        emit Withdrawn(msg.sender, amount);
    }
```

```
    function exit() external {
        withdraw(balanceOf(msg.sender));
        getReward();
    }

    function getReward() public updateReward(msg.sender)    checkStart{
        uint256 reward = earned(msg.sender);
        if (reward > 0) {
            rewards[msg.sender] = 0;
            cock.safeTransfer(msg.sender, reward.mul(90).div(100));
            emit RewardPaid(msg.sender, reward);
              //社区奖励
              cock.safeTransfer(addrCBO,reward.mul(10).div(100));//knownsec// 10%的社区奖励
        }
    }

   // modifier checkhalve(){
    //     if (block.timestamp >= periodFinish) {
    //         initreward = initreward.mul(50).div(100);
    //         cock.mint(address(this),initreward);

    //         rewardRate = initreward.div(DURATION);
    //         periodFinish = block.timestamp.add(DURATION);
    //         emit RewardAdded(initreward);
    //     }
    //     _;
    // }


        //开采结束算法
    modifier checkEnd(){
        require(periodFinish >= starttime,"ended");
        _;
    }

     function initContract(address community) public onlyOwner{//knownsec//  合约初始化,仅 owner 调用
        require(initialized == 0, " contract has initialized");
        addrCBO = community;
        cock.mint(address(this),initreward);
        rewardRate = initreward.div(DURATION);
        periodFinish = starttime.add(DURATION);
        initialized = 1;
        emit RewardAdded(initreward);
     }

    modifier checkStart(){
        require(block.timestamp > starttime,"not start");
        _;
    }

    function notifyRewardAmount(uint256 reward)
        external
        onlyRewardDistribution
        updateReward(address(0))
    {
        if (block.timestamp >= periodFinish) {
            rewardRate = reward.div(DURATION);
        } else {
            uint256 remaining = periodFinish.sub(block.timestamp);
            uint256 leftover = remaining.mul(rewardRate);
            rewardRate = reward.add(leftover).div(DURATION);
        }
        cock.mint(address(this),reward);
        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp.add(DURATION);
        emit RewardAdded(reward);
    }
}
```

TrxPool.sol

```
pragma solidity ^0.5.0;

/**
 * @dev Standard math utilities missing in the Solidity language.
 */
library Math {
    /**
     * @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, so we distribute
        return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
    }
}

// File: @openzeppelin/contracts/math/SafeMath.sol

pragma solidity ^0.5.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when 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.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0._
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
```

```
     * Requirements:
     * - Multiplication cannot overflow.
     */
    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;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    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;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}
```

```
// File: @openzeppelin/contracts/GSN/Context.sol

pragma solidity ^0.5.0;

/*
 * @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 GSN 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.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

// File: @openzeppelin/contracts/ownership/Ownable.sol

pragma solidity ^0.5.0;

/**
 * @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.
 *
 * 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.
 */
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 () internal {
        _owner = _msgSender();
        emit OwnershipTransferred(address(0), _owner);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return _msgSender() == _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 onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = 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 onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol

pragma solidity ^0.5.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
interface IERC20 {
    /**
     * @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 `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);
    function mint(address account, uint amount) external;

    /**
     * @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 `sender` to `recipient` 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 sender, address recipient, uint256 amount) external returns (bool);
```

```
    /**
     * @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);
}

// File: @openzeppelin/contracts/utils/Address.sol

pragma solidity ^0.5.5;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * This test is non-exhaustive, and there may be false-negatives: during the
     * execution of a contract's constructor, its address will be reported as
     * not containing a contract.
     *
     * IMPORTANT: It is unsafe to assume that an address for which this
     * function returns false is an externally-owned account (EOA) and not a
     * contract.
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        // bytes32 codehash;
        //bytes32                                   accountHash                                 =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        // assembly { codehash := extcodehash(account) }
        //return (codehash != 0x0 && codehash != accountHash);


        uint size;

        assembly { size := extcodesize(account) }

        return size > 0;
    }

    /**
     * @dev Converts an `address` into `address payable`. Note that this is
     * simply a type cast: the actual underlying value is not changed.
     *
     * _Available since v2.4.0._
     */
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-
pattern[checks-effects-interactions pattern].
     *
     * _Available since v2.4.0._
     */
```

```
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}

// File: @openzeppelin/contracts/token/ERC20/SafeERC20.sol

pragma solidity ^0.5.0;



/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for ERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased
allowance below zero");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing the requirement
     * on the return value: the return value is optional (but if data is returned, it must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism, since
        // we're implementing it ourselves.

        // A Solidity high level call has three parts:
        //   1. The target address is checked to verify it contains contract code
        //   2. The call itself is made, and success asserted
        //   3. The return value is decoded, which in turn checks the size of the returned data.
        // solhint-disable-next-line max-line-length
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}
```

```
// File: contracts/IRewardDistributionRecipient.sol

pragma solidity ^0.5.0;


contract IRewardDistributionRecipient is Ownable {
    //奖金分红地址
     address rewardDistribution;
     //通知奖励金额
    function notifyRewardAmount(uint256 reward) external;

    modifier onlyRewardDistribution() {
        require(_msgSender() == rewardDistribution, "Caller is not reward distribution");
        _;
    }
     //设置奖金分红地址
    function setRewardDistribution(address _rewardDistribution)
        external
        onlyOwner
    {
        rewardDistribution = _rewardDistribution;
    }
}

// File: contracts/CurveRewards.sol

pragma solidity ^0.5.0;


contract LPTokenWrapper {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;


    uint256 private _totalSupply;
    mapping(address => uint256) private _balances;


    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    function stake(uint256 amount) public payable {
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);
       //y.safeTransferFrom(msg.sender, address(this), amount);
    }

    function withdraw(uint256 amount) public {
         require(_balances[msg.sender].sub(amount)>=0,'withdraw amount error');
        _totalSupply = _totalSupply.sub(amount);
        _balances[msg.sender] = _balances[msg.sender].sub(amount);
       //y.safeTransfer(msg.sender, amount);
    }
}

contract TrxPoolReward is LPTokenWrapper, IRewardDistributionRecipient {

     IERC20 public cock = IERC20(0x412D6284C1EB4EBE9B1A14304560740 23A082D06C5);


     uint256 public constant DURATION = 4 weeks;
     uint8 public initialized = 0;
    uint256 public initreward = 1000*1e18;
    uint256 public starttime = 1600696800;
    uint256 public periodFinish = 0;
    uint256 public rewardRate = 0;
    uint256 public lastUpdateTime;
    uint256 public rewardPerTokenStored;
    mapping(address => uint256) public userRewardPerTokenPaid;
    mapping(address => uint256) public rewards;



     address public addrCBO;

    event RewardAdded(uint256 reward);
    event Staked(address indexed user, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);
    event RewardPaid(address indexed user, uint256 reward);
```

```
modifier updateReward(address account) {
        rewardPerTokenStored = rewardPerToken();
        lastUpdateTime = lastTimeRewardApplicable();
        if (account != address(0)) {
            rewards[account] = earned(account);
            userRewardPerTokenPaid[account] = rewardPerTokenStored;
        }
        _;
}

function lastTimeRewardApplicable() public view returns (uint256) {
        return Math.min(block.timestamp, periodFinish);
}

function rewardPerToken() public view returns (uint256) {
        if (totalSupply() == 0) {
            return rewardPerTokenStored;
        }

        return
            rewardPerTokenStored.add(
                lastTimeRewardApplicable()
                    .sub(lastUpdateTime)
                    .mul(rewardRate)
                    .mul(1e18)
                    .div(totalSupply())
            );
}

function earned(address account) public view returns (uint256) {
        return
            balanceOf(account)
                .mul(rewardPerToken().sub(userRewardPerTokenPaid[account]))
                .div(1e18)
                .add(rewards[account]);
}

function stake(uint256 amount) public payable updateReward(msg.sender)    checkStart checkEnd{
        require(amount > 0, "Cannot stake 0");
          require(amount == msg.value, "msg.value wrong");
        super.stake(amount);
        emit Staked(msg.sender, amount);

}

function withdraw(uint256 amount) public updateReward(msg.sender)    checkStart{
        require(amount > 0, "Cannot withdraw 0");
          super.withdraw(amount);
          msg.sender.transfer(amount);
        emit Withdrawn(msg.sender, amount);
}

function exit() external {
        withdraw(balanceOf(msg.sender));
        getReward();
}

function getReward() public updateReward(msg.sender)    checkStart{
        uint256 reward = earned(msg.sender);
        if (reward > 0) {
            rewards[msg.sender] = 0;
            cock.safeTransfer(msg.sender, reward.mul(90).div(100));
                cock.safeTransfer(addrCBO,reward.mul(10).div(100));
            emit RewardPaid(msg.sender, reward);
        }
}

modifier checkEnd(){
        require(periodFinish >= starttime,"ended");
        _;
}

 function initContract(address community) public onlyOwner{
        require(initialized == 0, " contract has initialized");
        addrCBO = community;
        cock.mint(address(this),initreward);
        rewardRate = initreward.div(DURATION);
        periodFinish = starttime.add(DURATION);
        initialized = 1;
        emit RewardAdded(initreward);
 }
```

```
    modifier checkStart(){
        require(block.timestamp > starttime,"not start");
          require(initialized == 1,"contract not init");
        _;
    }

    function notifyRewardAmount(uint256 reward)
        external
        onlyRewardDistribution
        updateReward(address(0))
          onlyOwner
    {
        if (block.timestamp >= periodFinish) {
            rewardRate = reward.div(DURATION);
        } else {
            uint256 remaining = periodFinish.sub(block.timestamp);
            uint256 leftover = remaining.mul(rewardRate);
            rewardRate = reward.add(leftover).div(DURATION);
        }
        cock.mint(address(this),reward);
        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp.add(DURATION);
        emit RewardAdded(reward);
    }

     function dev(address community) public onlyOwner{
        //require(msg.sender == addrCBO, "dev: wut?");
        addrCBO = community;
    }
}
```

UsdjPool.sol

```
pragma solidity ^0.5.0;

/**
  * @dev Standard math utilities missing in the Solidity language.
  */
library Math {
    /**
      * @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, so we distribute
        return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
    }
}

// File: @openzeppelin/contracts/math/SafeMath.sol

pragma solidity ^0.5.0;

/**
  * @dev Wrappers over Solidity's arithmetic operations with added overflow
  * checks.
  *
  * Arithmetic operations in Solidity wrap on overflow. This can easily result
  * in bugs, because programmers usually assume that an overflow raises an
  * error, which is the standard behavior in high level programming languages.
  * `SafeMath` restores this intuition by reverting the transaction when 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.
  */
library SafeMath {
```

```
/**
 * @dev Returns the addition of two unsigned integers, reverting on
 * overflow.
 *
 * Counterpart to Solidity's `+` operator.
 *
 * Requirements:
 * - Addition cannot overflow.
 */
function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");

    return c;
}

/**
 * @dev Returns the subtraction of two unsigned integers, reverting on
 * overflow (when the result is negative).
 *
 * Counterpart to Solidity's `-` operator.
 *
 * Requirements:
 * - Subtraction cannot overflow.
 */
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    return sub(a, b, "SafeMath: subtraction overflow");
}

/**
 * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
 * overflow (when the result is negative).
 *
 * Counterpart to Solidity's `-` operator.
 *
 * Requirements:
 * - Subtraction cannot overflow.
 *
 * _Available since v2.4.0._
 */
function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    require(b <= a, errorMessage);
    uint256 c = a - b;

    return c;
}

/**
 * @dev Returns the multiplication of two unsigned integers, reverting on
 * overflow.
 *
 * Counterpart to Solidity's `*` operator.
 *
 * Requirements:
 * - Multiplication cannot overflow.
 */
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;
}

/**
 * @dev Returns the integer division of two unsigned integers. Reverts on
 * division by zero. The result is rounded towards zero.
 *
 * Counterpart to Solidity's `/` operator. Note: this function uses a
 * `revert` opcode (which leaves remaining gas untouched) while Solidity
 * uses an invalid opcode to revert (consuming all remaining gas).
 *
 * Requirements:
 * - The divisor cannot be zero.
 */
function div(uint256 a, uint256 b) internal pure returns (uint256) {
    return div(a, b, "SafeMath: division by zero");
}

/**
 * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
```

```
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    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;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

// File: @openzeppelin/contracts/GSN/Context.sol

pragma solidity ^0.5.0;

/*
 * @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 GSN 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.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

// File: @openzeppelin/contracts/ownership/Ownable.sol
```

```
pragma solidity ^0.5.0;

/**
 * @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.
 *
 * 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.
 */
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 () internal {
        _owner = _msgSender();
        emit OwnershipTransferred(address(0), _owner);
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return _msgSender() == _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 onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = 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 onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol

pragma solidity ^0.5.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
interface IERC20 {
    /**
     * @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 `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);
    function mint(address account, uint amount) external;

    /**
     * @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 `sender` to `recipient` 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 sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @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);
}

// File: @openzeppelin/contracts/utils/Address.sol

pragma solidity ^0.5.5;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * This test is non-exhaustive, and there may be false-negatives: during the
     * execution of a contract's constructor, its address will be reported as
     * not containing a contract.
     *
     * IMPORTANT: It is unsafe to assume that an address for which this
     * function returns false is an externally-owned account (EOA) and not a
     * contract.
```

```
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        // bytes32 codehash;
        //bytes32                                                   accountHash                                                 =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        // assembly { codehash := extcodehash(account) }
        //return (codehash != 0x0 && codehash != accountHash);


        uint size;

        assembly { size := extcodesize(account) }

        return size > 0;
    }

    /**
     * @dev Converts an `address` into `address payable`. Note that this is
     * simply a type cast: the actual underlying value is not changed.
     *
     * _Available since v2.4.0._
     */
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-
pattern[checks-effects-interactions pattern].
     *
     * _Available since v2.4.0._
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}

// File: @openzeppelin/contracts/token/ERC20/SafeERC20.sol

pragma solidity ^0.5.0;



/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for ERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
```

```
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased
allowance below zero");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing the requirement
     * on the return value: the return value is optional (but if data is returned, it must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism, since
        // we're implementing it ourselves.

        // A Solidity high level call has three parts:
        //   1. The target address is checked to verify it contains contract code
        //   2. The call itself is made, and success asserted
        //   3. The return value is decoded, which in turn checks the size of the returned data.
        // solhint-disable-next-line max-line-length
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

// File: contracts/IRewardDistributionRecipient.sol

pragma solidity ^0.5.0;


contract IRewardDistributionRecipient is Ownable {
    address rewardDistribution;
    function notifyRewardAmount(uint256 reward) external;

    modifier onlyRewardDistribution() {
        require(_msgSender() == rewardDistribution, "Caller is not reward distribution");
        _;
    }
    function setRewardDistribution(address _rewardDistribution)
        external
        onlyOwner
    {
        rewardDistribution = _rewardDistribution;
    }
}

// File: contracts/CurveRewards.sol

pragma solidity ^0.5.0;


contract LPTokenWrapper {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    IERC20 public y = IERC20(0x41834295921A488D9D42B4B3021ED1A3C39FB0F03E); //usdj:
```

```
    uint256 private _totalSupply;

    mapping(address => uint256) private _balances;


    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    function stake(uint256 amount) public {
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);
        y.safeTransferFrom(msg.sender, address(this), amount);
    }
    function withdraw(uint256 amount) public {
         require(_balances[msg.sender].sub(amount)>=0,'withdraw amount error');
        _totalSupply = _totalSupply.sub(amount);
        _balances[msg.sender] = _balances[msg.sender].sub(amount);
        y.transfer(msg.sender,amount);
    }
}

contract UsujPoolReward is LPTokenWrapper, IRewardDistributionRecipient {


     IERC20 public cock = IERC20(0x412D6284C1EB4EBE9B1A14304560740 23A082D06C5);
     uint256 public constant DURATION = 4 weeks;
     uint8 public initialized = 0;
    uint256 public initreward = 1500*1e18;
    uint256 public starttime = 1600696800;
    uint256 public periodFinish = 0;
    uint256 public rewardRate = 0;
    uint256 public lastUpdateTime;
    uint256 public rewardPerTokenStored;
    mapping(address => uint256) public userRewardPerTokenPaid;
    mapping(address => uint256) public rewards;


     address public addrCBO;

    event RewardAdded(uint256 reward);
    event Staked(address indexed user, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);
    event RewardPaid(address indexed user, uint256 reward);


    modifier updateReward(address account) {

        rewardPerTokenStored = rewardPerToken();

        lastUpdateTime = lastTimeRewardApplicable();
        if (account != address(0)) {
            rewards[account] = earned(account);
            userRewardPerTokenPaid[account] = rewardPerTokenStored;
        }
        _;
    }

    function lastTimeRewardApplicable() public view returns (uint256) {
        return Math.min(block.timestamp, periodFinish);
    }


    function rewardPerToken() public view returns (uint256) {
          if (totalSupply() == 0) {
            return rewardPerTokenStored;
        }
          return
            rewardPerTokenStored.add(
                lastTimeRewardApplicable()
                    .sub(lastUpdateTime)
                    .mul(rewardRate)
                    .mul(1e18)
                    .div(totalSupply())
            );
    }

    function earned(address account) public view returns (uint256) {
        return
            balanceOf(account)
                .mul(rewardPerToken().sub(userRewardPerTokenPaid[account]))
                .div(1e18)
```

```
                .add(rewards[account]);
    }

    // stake visibility is public as overriding LPTokenWrapper's stake() function

    function stake(uint256 amount) public updateReward(msg.sender)    checkStart checkEnd{
        require(amount > 0, "Cannot stake 0");
        super.stake(amount);
        emit Staked(msg.sender, amount);
    }

    function withdraw(uint256 amount) public updateReward(msg.sender)    checkStart{
        require(amount > 0, "Cannot withdraw 0");
          super.withdraw(amount);
        emit Withdrawn(msg.sender, amount);
    }

    function exit() external {
        withdraw(balanceOf(msg.sender));
        getReward();
    }

    function getReward() public updateReward(msg.sender)    checkStart{
        uint256 reward = earned(msg.sender);
        if (reward > 0) {
            rewards[msg.sender] = 0;
            cock.safeTransfer(msg.sender, reward.mul(90).div(100));
               cock.safeTransfer(addrCBO,reward.mul(10).div(100));
            emit RewardPaid(msg.sender, reward);
        }
    }

    modifier checkEnd(){
          require(periodFinish >= starttime,"ended");
        _;
    }

     function initContract(address community) public onlyOwner{
          require(initialized == 0, " contract has initialized");
          addrCBO = community;
          cock.mint(address(this),initreward);
          rewardRate = initreward.div(DURATION);
          periodFinish = starttime.add(DURATION);
          initialized = 1;
          emit RewardAdded(initreward);
     }

    modifier checkStart(){
        require(block.timestamp > starttime,"not start");
          require(initialized == 1,"contract not init");
        _;
    }

    function notifyRewardAmount(uint256 reward)
        external
        onlyRewardDistribution
        updateReward(address(0))
          onlyOwner
    {
        if (block.timestamp >= periodFinish) {
            rewardRate = reward.div(DURATION);
        } else {
            uint256 remaining = periodFinish.sub(block.timestamp);
            uint256 leftover = remaining.mul(rewardRate);
            rewardRate = reward.add(leftover).div(DURATION);
        }
        cock.mint(address(this),reward);
        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp.add(DURATION);
        emit RewardAdded(reward);
    }

     function dev(address community) public onlyOwner{
        //require(msg.sender == addrCBO, "dev: wut?");
        addrCBO = community;
    }
}
```

6. 附录 B：安全风险评级标准

智能合约漏洞评级标准	
漏洞评级	**漏洞评级说明**
高危漏洞	能直接造成代币合约或用户资金损失的漏洞，如：能造成代币价值归零的数值溢出漏洞、能造成交易所损失代币的假充值漏洞、能造成合约账户损失 TRX 或代币的重入漏洞等； 能造成代币合约归属权丢失的漏洞，如：关键函数的访问控制缺陷、call 注入导致关键函数访问控制绕过等； 能造成代币合约无法正常工作的漏洞，如：因向恶意地址发送 TRX 导致的拒绝服务漏洞、因 energy 耗尽导致的拒绝服务漏洞。
中危漏洞	需要特定地址才能触发的高风险漏洞，如代币合约拥有者才能触发的数值溢出漏洞等；非关键函数的访问控制缺陷、不能造成直接资金损失的逻辑设计缺陷等。
低危漏洞	难以被触发的漏洞、触发之后危害有限的漏洞，如需要大量 TRX 或代币才能触发的数值溢出漏洞、触发数值溢出后攻击者无法直接获利的漏洞、通过指定高 energy 触发的事务顺序依赖风险等。

7. 附录 C：智能合约安全审计工具简介

7.1 Manticore

Manticore 是一个分析二进制文件和智能合约的符号执行工具，Manticore 包含一个符号虚拟机（EVM），一个 EVM 反汇编器/汇编器以及一个用于自动编译和分析 Solidity 的方便界面。它还集成了 Ethersplay，用于 EVM 字节码的 Bit of Traits of Bits 可视化反汇编程序，用于可视化分析。与二进制文件一样，Manticore 提供了一个简单的命令行界面和一个用于分析 EVM 字节码的 Python API。

7.2 Oyente

Oyente 是一个智能合约分析工具，Oyente 可以用来检测智能合约中常见的 bug，比如 reentrancy、事务排序依赖等等。更方便的是，Oyente 的设计是模块化的，所以这让高级用户可以实现并插入他们自己的检测逻辑，以检查他们的合约中自定义的属性。

7.3 securify.sh

Securify 可以验证智能合约常见的安全问题，例如交易乱序和缺少输入验证，它在全自动化的同时分析程序所有可能的执行路径，此外，Securify 还具有用于指定漏洞的特定语言，这使 Securify 能够随时关注当前的安全性和其他可靠性问题。

7.4 Echidna

Echidna 是一个为了对 EVM 代码进行模糊测试而设计的 Haskell 库。

7.5 MAIAN

MAIAN 是一个用于查找智能合约漏洞的自动化工具，Maian 处理合约的字

节码，并尝试建立一系列交易以找出并确认错误。

7.6 ethersplay

ethersplay 是一个 EVM 反汇编器，其中包含了相关分析工具。

7.7 ida-evm

ida-evm 是一个针对虚拟机（EVM）的 IDA 处理器模块。

7.8 Remix-ide

Remix 是一款基于浏览器的编译器和 IDE，可让用户使用 Solidity 语言构建合约并调试交易。

7.9 知道创宇区块链安全审计人员专用工具包

知道创宇安全审计人员专用工具包，由知道创宇渗透测试工程师研发，收集和使用，包含专用于测试人员的批量自动测试工具，自主研发的工具、脚本或利用工具等。

知道创宇

北京知道创宇信息技术股份有限公司

咨询电话 +86(10)400 060 9587

邮　　箱 sec@knownsec.com

官　　网 www.knownsec.com

地　　址 北京市 朝阳区 望京 SOHO T2-B座-2509