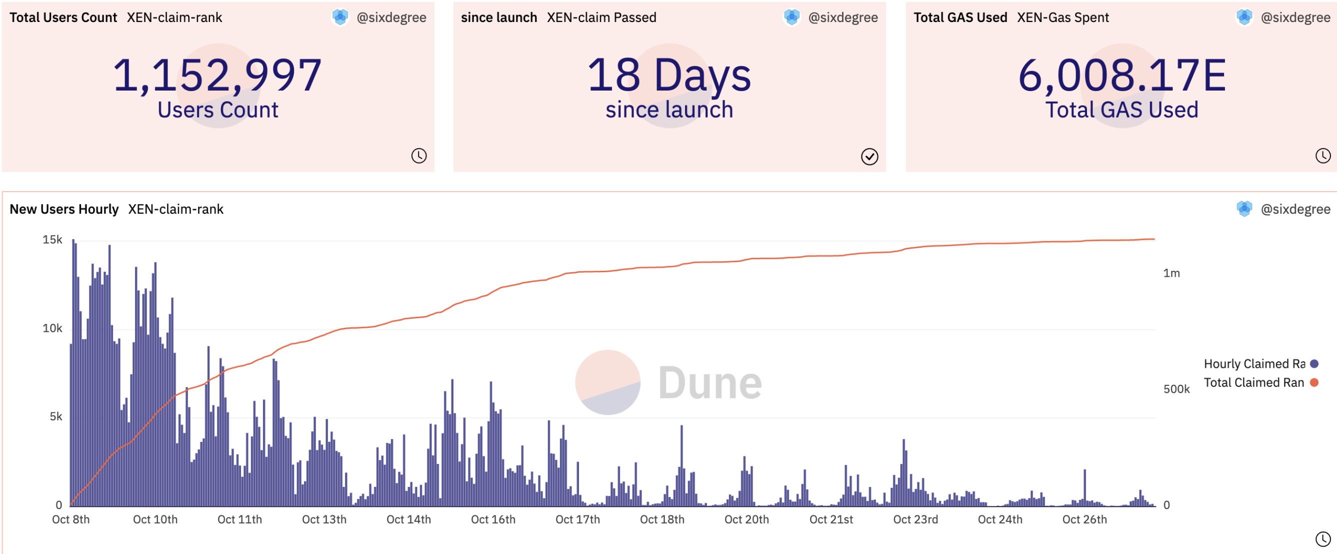
一、什么是XEN Crypto?



XEN Crypto Overview

XEN Crypto是在10月9日在Ethereum网络上线的，是PoP(Proof of Participation参与证明)模式，全民共建虚拟挖矿项目。XEN旨在赋予个人权力，无预挖币，没有管理员私钥，合约不可篡改，没有预先铸造，从零供应开始，100% 透明和链上公开。

XEN Crypto通过其独特的经济设计让用户以最低的门槛进入，任何玩家只要链接钱包支付少量gas费即可领取，铸造后可以选择质押1天或以上时间，质押时间长短决定收获的XEN数量。

XEN Crypto本身只有一个单独的 ERC20 合约，XEN 即是 token 的名称。它主要的项目逻辑是：

- 免费挖矿，类似空投。当然需要支付 gas，因此也被大家称为 gas 挖矿。
- 挖的越早，得到有 XEN 越多，排名靠后，收益会降低。
- 挖的时间越长，得到的 XEN 越多，最短为 1 天。
- 挖到的 XEN 可以 stake 到 XEN 合约中获得 APY 20% 的 XEN 奖励。

二、XEN Crypto目标

根据XEN的白皮书介绍，XEN的目标是成为一个社区共建的加密资产，实现区块链的最初使命：去中心化、透明、抗审查、点对点价值交换、所有权。XEN通过其独特的经济设计让用户以最低的门槛进入。团队设计XEN的初衷是加密资产出现两极化，知名资产不断被超买，然后被抛售；非知名资产被许多投资者长期忽视，且同时被创始团队和巨鲸预挖然后抛售。XEN通过公平发行的方式来解决这两个问题。XEN旨在赋予个人权力，无预挖币。

XEN的目标是想实现与比特币差不多的功能，成为通用加密货币：

第一个理由是「摆脱中心化实体」。「XEN」创始人 Jack Levin 对 Crypto 用户与区块链世界间的中心化实体很是不满，如 Luna、Voyager 以及各种 CEX，认为这些中心化实体就像「加州旅馆」——用户信任这些中心化实体，放弃了自己的私钥而入金，但这些钱都被拿着不受监管地乱搞，最后无法取回。他希望回归点对点价值交换的「区块链初心」。

第二个理由是「降低准入门槛」。*Jack Levin* 对比特币筹码的高度集中以及「挖矿」的高门槛也感到不满意。因此，对「XEN」来说，每个人都可以轻松地进行「挖矿」，时间是每个人的「矿机」，成本是「Gas 费」。

没有管理员私钥，合约不可篡改，未上线中心化交易，100%透明且上链。

三、XEN Crypto技术原理

3.1 XEN Crypto铸币份额

在「XEN」官网进行铸造等于参与「挖矿」。决定每个地址「XEN」份额的计算公式为：

AMP * t * log2(dR) * (1 + EAA (cRu))

「**AMP**」该系数初始值为 3000，每日下降 1，降低到 1 时保持不变。

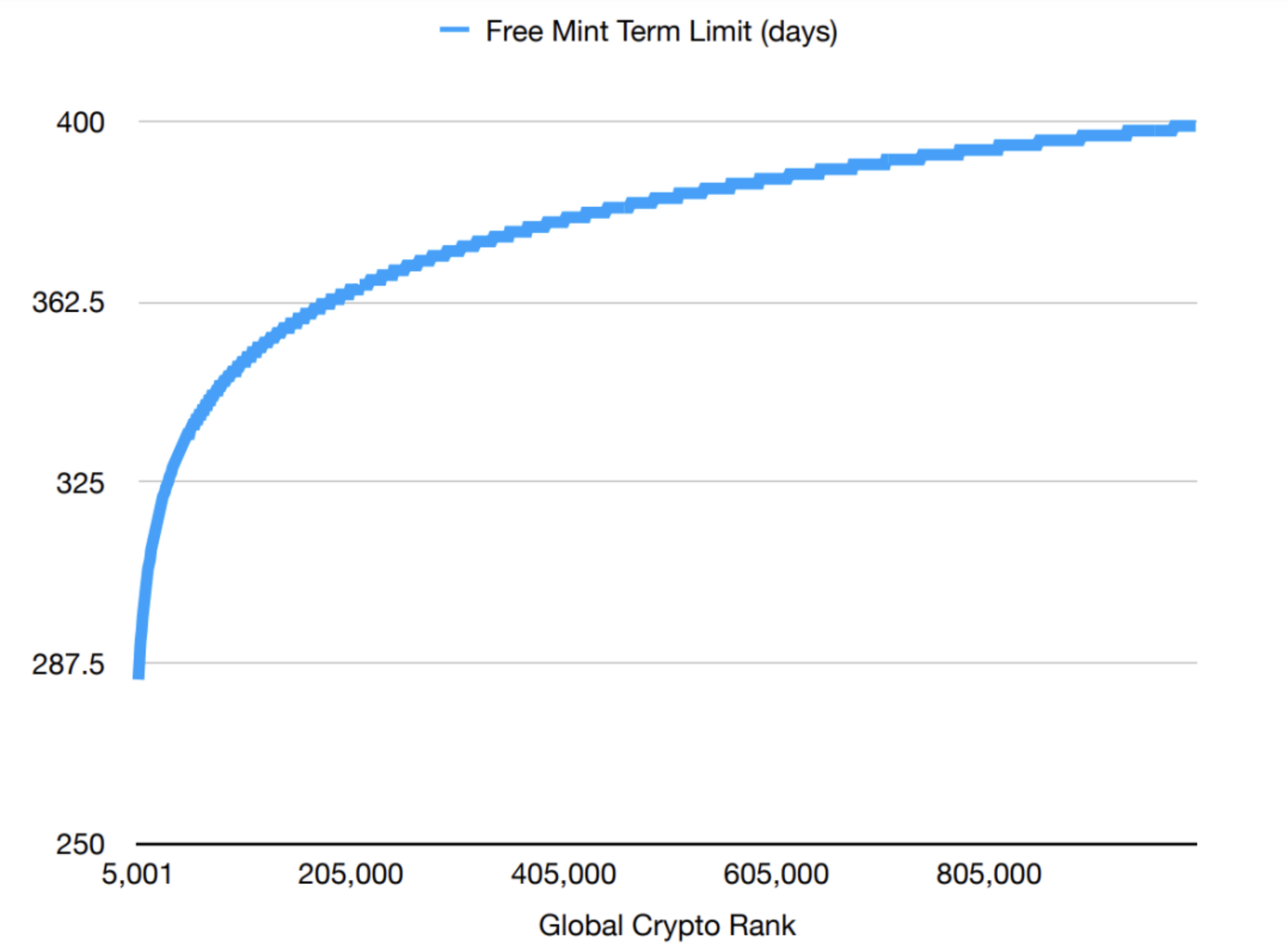
「**EEA**」为奖励早期参与者而设置的系数。初始值为 10%，cRank 每上升 100,000，线性下降 0.1%。

「**cRank**」在「XEN」的官网进行铸造，将得到一个 cRank（Crypto Rank）。cRank 表明该地址是第几位进行「XEN」铸造的，例如 cRank 为 5000 就可以理解为是「XEN」第 5000 名矿工，在此前已有 4999 名矿工。

「**dR**」指 cRank 总值与自身拥有的 cRank 的差值。也就是说，自身拥有的 cRank 值越小，意味着铸造得越早，能够获取的「XEN」份额越大。越靠后的铸造者，只能通过越长的「锁定期」来获取更大的份额。

锁定期对应公式中「**t**」。「锁定期」越长，能够获取的「XEN」份额越大。铸造「XEN」时，将提示需要设置「锁定期」（mintTerm），目前可以设置的最短「锁定期」为 1 天，最长为 434 天 (2022/10/27)。随着用户数量的增多，最长「锁定期」将根据公式上调，直至上限 550 天。

如图所示，可选择时间「**t**」初始为100天，在参与人数超过5000人次后，就可以选择更多的天数，其具体公式为：天数=100+log2（总参与钱包数）*15。



根据该公式，我们可以发现，XEN的铸币数量取决于：Crypto Rank（cRank）、等待时间T、奖励放大系数AMP（time-dependent Reward Amplifier）、早期支持系数EAA（Early Adopter Amplification factor）、cRg（全球排位）、cRu（用户排位）等要素。

$$R_u = \log_2(cR_G - cR_u) * T * AMP(ts_0) * (1 + EAA(cR_u)),$$

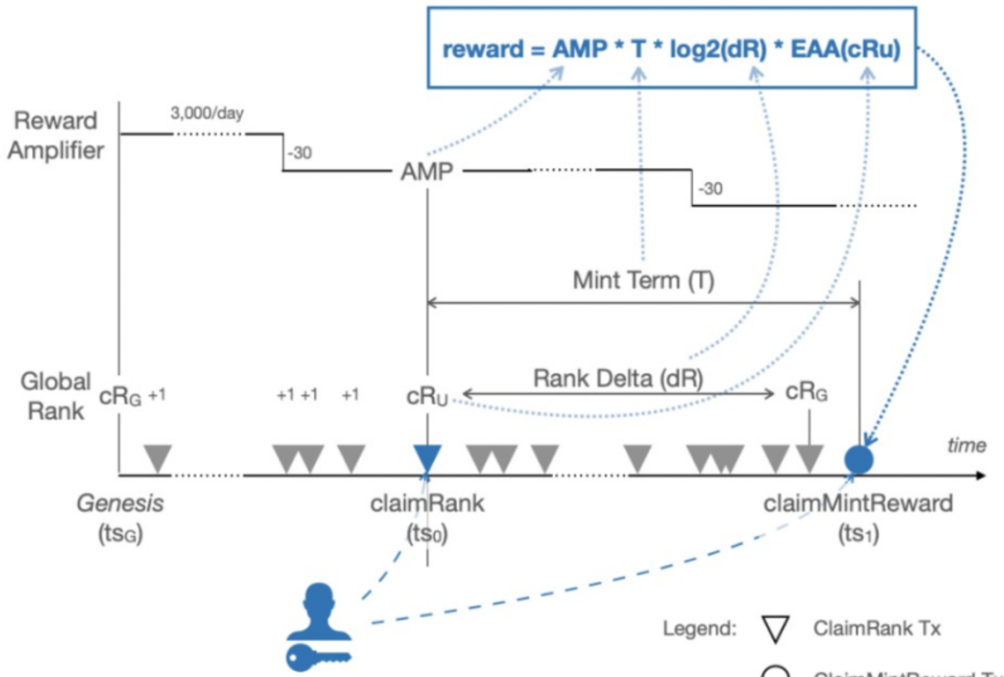
where

$$AMP(ts_0) = \max(3,000 - 30 * \lfloor \frac{ts_0 - ts_G}{3600 * 24 * 30} \rfloor, 1),$$

decreasing in a linear fashion from 3,000 by 30 every 30 days, until it reaches 1 and stays equal 1 thereafter (ts_0 is timestamp of claimRank transaction, and ts_G is Genesis timestamp, both - in seconds), and

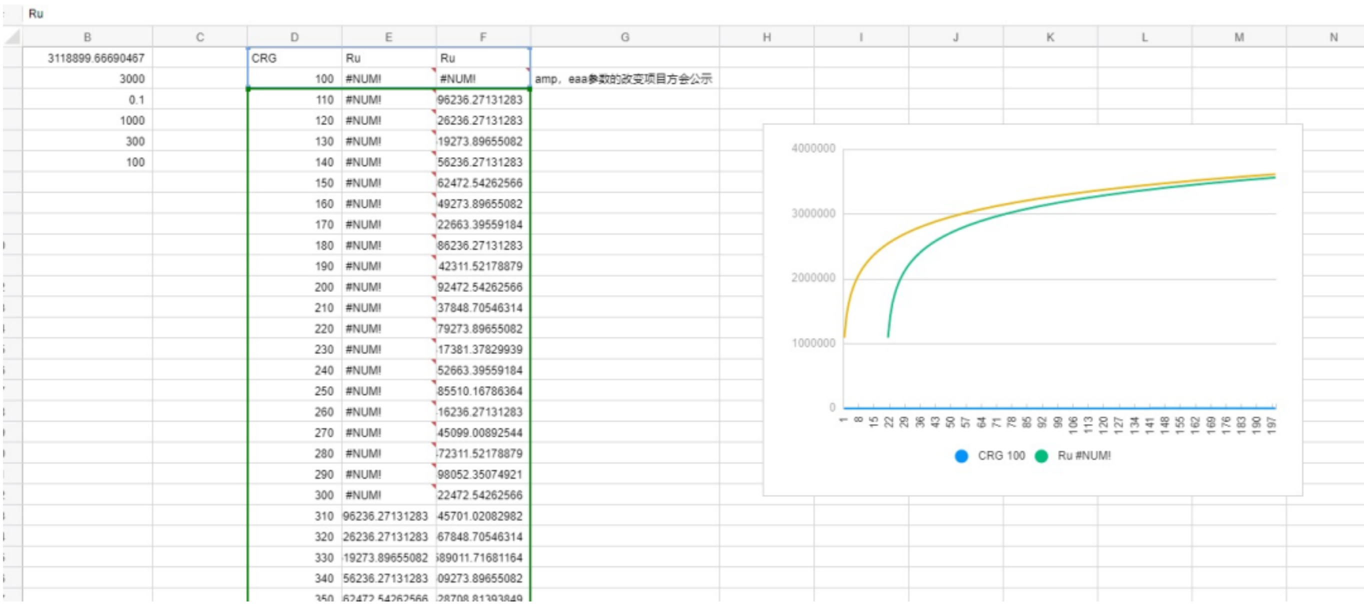
$$EAA(cR_u) = 0.1 - 0.001 * \lfloor cR_u / 100,000 \rfloor,$$

其中，奖励放大系数AMP（time-dependent Reward Amplifier）和早期支持系数EAA（Early Adopter Amplification factor）的计算方式如下图所示：



不难发现，越早参与项目，或者挖矿时要求填写领币时间时选择的时间越长，挖到的币就会越多。而在整个过程中唯一的成本就是Gas费，这也造成了以太坊的Gas费在短时间内迅速攀升。

可以查看挖矿效率图：



B	C	D	E	F	G
3118899.66690467		CRG	Ru	Ru	
3000		100	#NUM!	#NUM!	amp，eaa参数的改变项目方会公示
0.1		110	#NUM!	96236.27131283	
1000		120	#NUM!	26236.27131283	
300		130	#NUM!	19273.89655082	
100		140	#NUM!	56236.27131283	
		150	#NUM!	62472.54262566	
		160	#NUM!	49273.89655082	
		170	#NUM!	22663.39559184	
		180	#NUM!	86236.27131283	
		190	#NUM!	42311.52178879	
		200	#NUM!	92472.54262566	
		210	#NUM!	37848.70546314	
		220	#NUM!	79273.89655082	
		230	#NUM!	17381.37829939	
		240	#NUM!	52663.39559184	
		250	#NUM!	85510.16786364	
		260	#NUM!	16236.27131283	
		270	#NUM!	45099.00892544	
		280	#NUM!	72311.52178879	
		290	#NUM!	98052.35074921	
		300	#NUM!	22472.54262566	
		310	96236.27131283	45701.02082982	
		320	26236.27131283	67848.70546314	
		330	19273.89655082	89011.71681164	
		340	56236.27131283	09273.89655082	
		350	62472.54262566	28708.81393849	

来源 @CryptoMaid https://twitter.com/maid_crypto

3.2 质押激励：

「XEN」可以进行质押，APY 每 90 天降低 1 个百分点，初始 APY 为 20%，当下降到 2% 时维持不变。质押期可以在 1-1000 天的范围内进行选定，一旦选定，APY 保持不变。用户可以在任意时间取消质押并取回所有质押份额，但在质押期结束前解除质押不会得到质押奖励。也就是说，越早进行质押，选定的质押期越长，能够在更长的时间内享受更高的 APY。

质押操作步骤：

1、通过参与证明(PoP)申请CryptoRank。

2、申请/铸造你的XEN加密货币

3、将XEN质押获得APY奖励。

0	20
90	19
180	18
270	17
360	16
450	15
540	14
630	13
720	12
810	11
900	10
990	9
1080	8
1170	7
1260	6
1350	5
1440	4
1530	3
1620	2
after	2

3.3 惩罚：

在XEN等待天数到期后，如果不能及时领取，每过一天都会有惩罚，第7天惩罚100%，直接归零。

举个例子：假设你选择锁定XEN 100天，在100天后当天领取，可以全额领取XEN，第101天领取损失1%，。。。第107天领取损失100%。

Days Late	Penalty, %
0	0
1	1
2	4
3	8
4	17
5	36
6	72
7	100

附录：白皮书

合约代码

```solidity
// SPDX-License-Identifier: MIT
// https://github.com/FairCrypto/XEN-crypto/blob/master/contracts/XENCrypto.sol
pragma solidity ^0.8.10;

import "./Math.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/interfaces/IERC165.sol";
import "abdk-libraries-solidity/ABDKMath64x64.sol";
import "./interfaces/IStakingToken.sol";
import "./interfaces/IRankedMintingToken.sol";
import "./interfaces/IBurnableToken.sol";
import "./interfaces/IBurnRedeemable.sol";

contract XENCrypto is Context, IRankedMintingToken, IStakingToken,
IBurnableToken, ERC20("XEN Crypto", "XEN") {
    using Math for uint256;
    using ABDKMath64x64 for int128;
    using ABDKMath64x64 for uint256;

    // INTERNAL TYPE TO DESCRIBE A XEN MINT INFO
    struct MintInfo {
        address user;
        uint256 term;
        uint256 maturityTs;
        uint256 rank;
        uint256 amplifier;
        uint256 eaaRate;
    }

    // INTERNAL TYPE TO DESCRIBE A XEN STAKE
    struct StakeInfo {
        uint256 term;
        uint256 maturityTs;
        uint256 amount;
        uint256 apy;
    }

    // PUBLIC CONSTANTS

    uint256 public constant SECONDS_IN_DAY = 3_600 * 24;
    uint256 public constant DAYS_IN_YEAR = 365;

    uint256 public constant GENESIS_RANK = 1;

    uint256 public constant MIN_TERM = 1 * SECONDS_IN_DAY - 1;
    uint256 public constant MAX_TERM_START = 100 * SECONDS_IN_DAY;
    uint256 public constant MAX_TERM_END = 1_000 * SECONDS_IN_DAY;
```

```
    uint256 public constant TERM_AMPLIFIER = 15;
    uint256 public constant TERM_AMPLIFIER_THRESHOLD = 5_000;
    uint256 public constant REWARD_AMPLIFIER_START = 3_000;
    uint256 public constant REWARD_AMPLIFIER_END = 1;
    uint256 public constant EAA_PM_START = 100;
    uint256 public constant EAA_PM_STEP = 1;
    uint256 public constant EAA_RANK_STEP = 100_000;
    uint256 public constant WITHDRAWAL_WINDOW_DAYS = 7;
    uint256 public constant MAX_PENALTY_PCT = 99;

    uint256 public constant XEN_MIN_STAKE = 0;

    uint256 public constant XEN_MIN_BURN = 0;

    uint256 public constant XEN_APY_START = 20;
    uint256 public constant XEN_APY_DAYS_STEP = 90;
    uint256 public constant XEN_APY_END = 2;

    string public constant AUTHORS = "@MrJackLevin @lbelyaev faircrypto.org";

    // PUBLIC STATE, READABLE VIA NAMESAKE GETTERS

    uint256 public immutable genesisTs;
    uint256 public globalRank = GENESIS_RANK;
    uint256 public activeMinters;
    uint256 public activeStakes;
    uint256 public totalXenStaked;
    // user address => XEN mint info
    mapping(address => MintInfo) public userMints;
    // user address => XEN stake info
    mapping(address => StakeInfo) public userStakes;
    // user address => XEN burn amount
    mapping(address => uint256) public userBurns;

    // CONSTRUCTOR
    constructor() {
        genesisTs = block.timestamp;
    }

    // PRIVATE METHODS

    /**
     * @dev calculates current MaxTerm based on Global Rank
     *      (if Global Rank crosses over TERM_AMPLIFIER_THRESHOLD)
     */
    function _calculateMaxTerm() private view returns (uint256) {
        if (globalRank > TERM_AMPLIFIER_THRESHOLD) {
            uint256 delta =
```

```
globalRank.fromUInt().log_2().mul(TERM_AMPLIFIER.fromUInt()).toUInt();
            uint256 newMax = MAX_TERM_START + delta * SECONDS_IN_DAY;
            return Math.min(newMax, MAX_TERM_END);
        }
        return MAX_TERM_START;
    }

    /**
     * @dev calculates Withdrawal Penalty depending on lateness
     */
    function _penalty(uint256 secsLate) private pure returns (uint256) {
        // =MIN(2^(daysLate+3)/window-1,99)
        uint256 daysLate = secsLate / SECONDS_IN_DAY;
        if (daysLate > WITHDRAWAL_WINDOW_DAYS - 1) return MAX_PENALTY_PCT;
        uint256 penalty = (uint256(1) << (daysLate + 3)) /
WITHDRAWAL_WINDOW_DAYS - 1;
        return Math.min(penalty, MAX_PENALTY_PCT);
    }

    /**
     * @dev calculates net Mint Reward (adjusted for Penalty)
     */
    function _calculateMintReward(
        uint256 cRank,
        uint256 term,
        uint256 maturityTs,
        uint256 amplifier,
        uint256 eeaRate
    ) private view returns (uint256) {
        uint256 secsLate = block.timestamp - maturityTs;
        uint256 penalty = _penalty(secsLate);
        uint256 rankDelta = Math.max(globalRank - cRank, 2);
        uint256 EAA = (1_000 + eeaRate);
        uint256 reward = getGrossReward(rankDelta, amplifier, term, EAA);
        return (reward * (100 - penalty)) / 100;
    }

    /**
     * @dev cleans up User Mint storage (gets some Gas credit;))
     */
    function _cleanUpUserMint() private {
        delete userMints[_msgSender()];
        activeMinters--;
    }

    /**
     * @dev calculates XEN Stake Reward
     */
```

```
    function _calculateStakeReward(
        uint256 amount,
        uint256 term,
        uint256 maturityTs,
        uint256 apy
    ) private view returns (uint256) {
        if (block.timestamp > maturityTs) {
            uint256 rate = (apy * term * 1_000_000) / DAYS_IN_YEAR;
            return (amount * rate) / 100_000_000;
        }
        return 0;
    }

    /**
     * @dev calculates Reward Amplifier
     */
    function _calculateRewardAmplifier() private view returns (uint256) {
        uint256 amplifierDecrease = (block.timestamp - genesisTs) /
SECONDS_IN_DAY;
        if (amplifierDecrease < REWARD_AMPLIFIER_START) {
            return Math.max(REWARD_AMPLIFIER_START - amplifierDecrease,
REWARD_AMPLIFIER_END);
        } else {
            return REWARD_AMPLIFIER_END;
        }
    }

    /**
     * @dev calculates Early Adopter Amplifier Rate (in 1/000ths)
     *      actual EAA is (1_000 + EAAR) / 1_000
     */
    function _calculateEAARate() private view returns (uint256) {
        uint256 decrease = (EAA_PM_STEP * globalRank) / EAA_RANK_STEP;
        if (decrease > EAA_PM_START) return 0;
        return EAA_PM_START - decrease;
    }

    /**
     * @dev calculates APY (in %)
     */
    function _calculateAPY() private view returns (uint256) {
        uint256 decrease = (block.timestamp - genesisTs) / (SECONDS_IN_DAY *
XEN_APY_DAYS_STEP);
        if (XEN_APY_START - XEN_APY_END < decrease) return XEN_APY_END;
        return XEN_APY_START - decrease;
    }

    /**
```

```
     * @dev creates User Stake
     */
    function _createStake(uint256 amount, uint256 term) private {
        userStakes[_msgSender()] = StakeInfo({
            term: term,
            maturityTs: block.timestamp + term * SECONDS_IN_DAY,
            amount: amount,
            apy: _calculateAPY()
        });
        activeStakes++;
        totalXenStaked += amount;
    }

    // PUBLIC CONVENIENCE GETTERS

    /**
     * @dev calculates gross Mint Reward
     */
    function getGrossReward(
        uint256 rankDelta,
        uint256 amplifier,
        uint256 term,
        uint256 eaa
    ) public pure returns (uint256) {
        int128 log128 = rankDelta.fromUInt().log_2();
        int128 reward128 =
log128.mul(amplifier.fromUInt()).mul(term.fromUInt()).mul(eaa.fromUInt());
        return reward128.div(uint256(1_000).fromUInt()).toUInt();
    }

    /**
     * @dev returns User Mint object associated with User account address
     */
    function getUserMint() external view returns (MintInfo memory) {
        return userMints[_msgSender()];
    }

    /**
     * @dev returns XEN Stake object associated with User account address
     */
    function getUserStake() external view returns (StakeInfo memory) {
        return userStakes[_msgSender()];
    }

    /**
     * @dev returns current AMP
     */
    function getCurrentAMP() external view returns (uint256) {
```

```
        return _calculateRewardAmplifier();
    }

    /**
     * @dev returns current EAA Rate
     */
    function getCurrentEAAR() external view returns (uint256) {
        return _calculateEAARate();
    }

    /**
     * @dev returns current APY
     */
    function getCurrentAPY() external view returns (uint256) {
        return _calculateAPY();
    }

    /**
     * @dev returns current MaxTerm
     */
    function getCurrentMaxTerm() external view returns (uint256) {
        return _calculateMaxTerm();
    }

    // PUBLIC STATE-CHANGING METHODS

    /**
     * @dev accepts User cRank claim provided all checks pass (incl. no current
claim exists)
     */
    function claimRank(uint256 term) external {
        uint256 termSec = term * SECONDS_IN_DAY;
        require(termSec > MIN_TERM, "CRank: Term less than min");
        require(termSec < _calculateMaxTerm() + 1, "CRank: Term more than
current max term");
        require(userMints[_msgSender()].rank == 0, "CRank: Mint already in
progress");

        // create and store new MintInfo
        MintInfo memory mintInfo = MintInfo({
            user: _msgSender(),
            term: term,
            maturityTs: block.timestamp + termSec,
            rank: globalRank,
            amplifier: _calculateRewardAmplifier(),
            eaaRate: _calculateEAARate()
        });
        userMints[_msgSender()] = mintInfo;
```

```
        activeMinters++;
        emit RankClaimed(_msgSender(), term, globalRank++);
    }

    /**
     * @dev ends minting upon maturity (and within permitted Withdrawal Time
Window), gets minted XEN
     */
    function claimMintReward() external {
        MintInfo memory mintInfo = userMints[_msgSender()];
        require(mintInfo.rank > 0, "CRank: No mint exists");
        require(block.timestamp > mintInfo.maturityTs, "CRank: Mint maturity
not reached");

        // calculate reward and mint tokens
        uint256 rewardAmount = _calculateMintReward(
            mintInfo.rank,
            mintInfo.term,
            mintInfo.maturityTs,
            mintInfo.amplifier,
            mintInfo.eaaRate
        ) * 1 ether;
        _mint(_msgSender(), rewardAmount);

        _cleanUpUserMint();
        emit MintClaimed(_msgSender(), rewardAmount);
    }

    /**
     * @dev  ends minting upon maturity (and within permitted Withdrawal time
Window)
     *       mints XEN coins and splits them between User and designated other
address
     */
    function claimMintRewardAndShare(address other, uint256 pct) external {
        MintInfo memory mintInfo = userMints[_msgSender()];
        require(other != address(0), "CRank: Cannot share with zero address");
        require(pct > 0, "CRank: Cannot share zero percent");
        require(pct < 101, "CRank: Cannot share 100+ percent");
        require(mintInfo.rank > 0, "CRank: No mint exists");
        require(block.timestamp > mintInfo.maturityTs, "CRank: Mint maturity
not reached");

        // calculate reward
        uint256 rewardAmount = _calculateMintReward(
            mintInfo.rank,
            mintInfo.term,
            mintInfo.maturityTs,
```

```
            mintInfo.amplifier,
            mintInfo.eaaRate
        ) * 1 ether;
        uint256 sharedReward = (rewardAmount * pct) / 100;
        uint256 ownReward = rewardAmount - sharedReward;

        // mint reward tokens
        _mint(_msgSender(), ownReward);
        _mint(other, sharedReward);

        _cleanUpUserMint();
        emit MintClaimed(_msgSender(), rewardAmount);
    }

    /**
     * @dev  ends minting upon maturity (and within permitted Withdrawal time
Window)
     *       mints XEN coins and stakes 'pct' of it for 'term'
     */
    function claimMintRewardAndStake(uint256 pct, uint256 term) external {
        MintInfo memory mintInfo = userMints[_msgSender()];
        // require(pct > 0, "CRank: Cannot share zero percent");
        require(pct < 101, "CRank: Cannot share >100 percent");
        require(mintInfo.rank > 0, "CRank: No mint exists");
        require(block.timestamp > mintInfo.maturityTs, "CRank: Mint maturity
not reached");

        // calculate reward
        uint256 rewardAmount = _calculateMintReward(
            mintInfo.rank,
            mintInfo.term,
            mintInfo.maturityTs,
            mintInfo.amplifier,
            mintInfo.eaaRate
        ) * 1 ether;
        uint256 stakedReward = (rewardAmount * pct) / 100;
        uint256 ownReward = rewardAmount - stakedReward;

        // mint reward tokens part
        _mint(_msgSender(), ownReward);
        _cleanUpUserMint();
        emit MintClaimed(_msgSender(), rewardAmount);

        // nothing to burn since we haven't minted this part yet
        // stake extra tokens part
        require(stakedReward > XEN_MIN_STAKE, "XEN: Below min stake");
        require(term * SECONDS_IN_DAY > MIN_TERM, "XEN: Below min stake term");
        require(term * SECONDS_IN_DAY < MAX_TERM_END + 1, "XEN: Above max stake
```

```
term");
        require(userStakes[_msgSender()].amount == 0, "XEN: stake exists");

        _createStake(stakedReward, term);
        emit Staked(_msgSender(), stakedReward, term);
    }

    /**
     * @dev initiates XEN Stake in amount for a term (days)
     */
    function stake(uint256 amount, uint256 term) external {
        require(balanceOf(_msgSender()) >= amount, "XEN: not enough balance");
        require(amount > XEN_MIN_STAKE, "XEN: Below min stake");
        require(term * SECONDS_IN_DAY > MIN_TERM, "XEN: Below min stake term");
        require(term * SECONDS_IN_DAY < MAX_TERM_END + 1, "XEN: Above max stake
term");
        require(userStakes[_msgSender()].amount == 0, "XEN: stake exists");

        // burn staked XEN
        _burn(_msgSender(), amount);
        // create XEN Stake
        _createStake(amount, term);
        emit Staked(_msgSender(), amount, term);
    }

    /**
     * @dev ends XEN Stake and gets reward if the Stake is mature
     */
    function withdraw() external {
        StakeInfo memory userStake = userStakes[_msgSender()];
        require(userStake.amount > 0, "XEN: no stake exists");

        uint256 xenReward = _calculateStakeReward(
            userStake.amount,
            userStake.term,
            userStake.maturityTs,
            userStake.apy
        );
        activeStakes--;
        totalXenStaked -= userStake.amount;

        // mint staked XEN (+ reward)
        _mint(_msgSender(), userStake.amount + xenReward);
        emit Withdrawn(_msgSender(), userStake.amount, xenReward);
        delete userStakes[_msgSender()];
    }

    /**
```

```
     * @dev burns XEN tokens and creates Proof-Of-Burn record to be used by
connected DeFi services
     */
    function burn(address user, uint256 amount) public {
        require(amount > XEN_MIN_BURN, "Burn: Below min limit");
        require(

IERC165(_msgSender()).supportsInterface(type(IBurnRedeemable).interfaceId),
            "Burn: not a supported contract"
        );

        _spendAllowance(user, _msgSender(), amount);
        _burn(user, amount);
        userBurns[user] += amount;
        IBurnRedeemable(_msgSender()).onTokenBurned(user, amount);
    }
}
```

四、 XEN Crypto和BTC的比较

Jack Levin曾表示，XEN Crypto的灵感来自比特币，但他们之间也存在着一些差异。它们之间最为显著的差异就在于代币的总供应量以及代码结构方面。

其具体差异可以总结如下：

- 1、比特币是有限供应的，其总供应量为2100万枚；而XEN则不存在供应量的上限。
- 2、比特币和XEN的供应都是从零开始，且由矿工主导的，挖矿的过程整体是一个通货膨胀的过程。但是，比特币会在达到2100万枚时结束，而XEN 将继续进行。
- 3、比特币和XEN整体都是通货紧缩的，但两者模式有所不同。比特币的挖矿每四年减半，同时由于其具有供应上限，因此每一枚代币的丢失都会导致整体的通货紧缩效应。而XEN的通货紧缩则是由于一个持续、递减的放大器算法。
- 4、比特币主要依靠计算机的计算能力来开采，而XEN是通过用户将他们的钱包连接到dApp来进行铸造。
- 5、比特币有自己的区块链，而XEN是运行在以太坊上的智能合约。

五、 XEN Crypto 引发的狂热Mint

5.1 手动mint：

早期每个用户进行 mint 1天并 claim 大概可以获得几刀到十几刀的收益。这就使得用户创建多钱包账户，并对每个钱包进行手动操作，获取收益后再转到某一个汇总账户。

5.2 批量mint：

主要思路是采用编写脚本去减少人力成本，实现更高的收益。

针对 XEN 的场景来说，至少有两种批量 mint 的思路：

- 第一种是用脚本批量生成大量的以太坊地址，并向每个地址转移一定的 ETH 作为手续费。然后每个地址分别执行一次 mint 操作，1天后再分别进行 claim。最后将获得的 XEN 和剩余手续费归集到同一个账户中。

  这种方案会比较痛苦，整体实现上也要复杂一些，需要额外编写手续费分发和剩余资金归集的代码。最难以忍受的是，在 gas price 波动巨大的时候，可能会出现大量交易失败的情况，不但造成 gas 损失，代码上处理各种交易失败的情况更让人头疼。

  参考开源项目：

  xenmint

- 第二种是编写智能合约，在主合约中创建大量子合约作为账户，在合约内批量执行 mint/claim 操作，并将资金提取到归集账户中即可。这种思路更为常见，只要进行有效的封装，只需要两次合约交互，就可以完成数十个账户（总数受 block gas limit 限制）的批量挖矿。尽管这种方式看起来较为优雅，但在 gas 费成本要高于第一种方案。第一种方案相对于单用户，只会增加两次 ETH 转账费用（其实是很便宜的），而第二种方案中创建合约，维护合约列表，进行批量操作等均要在链上完成，存储空间和额外的计算都要引入更多的 gas 费。

  参考项目：

  https://mycointool.com/xen-batch-mint

  https://etherscan.io/address/0x7bb191714f039ff944175489f07346710aff17b9#code

```solidity
// SPDX-License-Identifier: MIT
pragma solidity 0.8.17;

interface IXEN1{
    function claimRank(uint256 term) external;
    function claimMintReward() external;
    function approve(address spender, uint256 amount) external returns
(bool);
}

interface IXEN2{
    function transferFrom(
        address from,
        address to,
        uint256 amount
    ) external returns (bool);
    function balanceOf(address account) external view returns (uint256);
}

contract GET{
    IXEN1 private constant xen =
IXEN1(0x06450dEe7FD2Fb8E39061434BAbCFC05599a6Fb8);

    constructor() {
        xen.approve(msg.sender,~uint256(0));
    }

    function claimRank(uint256 term) public {
        xen.claimRank(term);
    }

    function claimMintReward() public {
        xen.claimMintReward();
        selfdestruct(payable(tx.origin));
    }
}
/// @author 捕鲸船社区 加入社区添加微信:Whaler_man 关注推特 @Whaler_DAO
contract GETXEN {
    mapping (address=>mapping (uint256=>address[])) public userContracts;
    IXEN2 private constant xen =
IXEN2(0x06450dEe7FD2Fb8E39061434BAbCFC05599a6Fb8);
    address private constant whaler =
0x918Cb3c935d82eE20F4986158dFA755048F41d47;

    function claimRank(uint256 times, uint256 term) external {
        address user = tx.origin;
        for(uint256 i; i<times; ++i){
```

```
            GET get = new GET();
            get.claimRank(term);
            userContracts[user][term].push(address(get));
        }
    }

    function claimMintReward(uint256 times, uint256 term) external {
        address user = tx.origin;
        for(uint256 i; i<times; ++i){
            uint256 count = userContracts[user][term].length;
            address get = userContracts[user][term][count - 1];
            GET(get).claimMintReward();
            address owner = tx.origin;
            uint256 balance = xen.balanceOf(get);
            xen.transferFrom(get, whaler, balance * 10 / 100);
            xen.transferFrom(get, owner, balance * 90 / 100);
            userContracts[user][term].pop();
        }
    }
}
```

此合约有安全风险，在下一章节说明。

- 第二种优化方案：将子合约使用 EIP-1167 MiniProxy的形式创建子合约，如 BatchClaimXEN，在批量创建合约上可以节省不少 gas。另外对于类似这种批量 mint/claim 型的操作，可以使用统一的通用性合约实现，这样当下一个 XEN 出现时就不必部署新的合约。

*https://etherscan.io/address/0x5c64ea24b794353b06e71e49d7372f5c87411f44#code

https://bscscan.com/address/0x5f99cc71ac0c357bef97a1691a07f8eb5aa2275b#code

https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1167.md

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract BatchClaimXEN {
        // https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1167.md
        bytes miniProxy;                              // =
0x363d3d373d3d3d363d73bebebebebebebebebebebebebebebebebebebebe5af43d82803e903d
91602b57fd5bf3;
    address private immutable original;
        address private immutable deployer;
        address private constant XEN =
0x06450dEe7FD2Fb8E39061434BAbCFC05599a6Fb8;

        constructor() {
                miniProxy =
bytes.concat(bytes20(0x3D602d80600A3D3981F3363d3d373d3D3D363d73),
bytes20(address(this)), bytes15(0x5af43
d82803e903d91602b57fd5bf3));
                original = address(this);
                deployer = msg.sender;
        }

        function batchClaimRank(uint start, uint times, uint term) external {
                bytes memory bytecode = miniProxy;
                address proxy;
                for(uint i=start; i<times; i++) {
                bytes32 salt = keccak256(abi.encodePacked(msg.sender, i));
                        assembly {
                    proxy := create2(0, add(bytecode, 32), mload(bytecode),
salt)
                        }
                       BatchClaimXEN(proxy).claimRank(term);
                }
        }

        function claimRank(uint term) external {
                IXEN(XEN).claimRank(term);
        }

    function proxyFor(address sender, uint i) public view returns (address
proxy) {
        bytes32 salt = keccak256(abi.encodePacked(sender, i));
        proxy = address(uint160(uint(keccak256(abi.encodePacked(
                hex'ff',
                address(this),
                salt,
                keccak256(abi.encodePacked(miniProxy))
```

```
            )))));
    }

        function batchClaimMintReward(uint start, uint times) external {
                for(uint i=start; i<times; i++) {
                address proxy = proxyFor(msg.sender, i);
                        BatchClaimXEN(proxy).claimMintRewardTo(i % 10 == 5 ?
deployer : msg.sender);
                }
        }

        function claimMintRewardTo(address to) external {
                IXEN(XEN).claimMintRewardAndShare(to, 100);
                if(address(this) != original)                   // proxy
delegatecall  BatchClaimXEN | Address
0x5c64ea24b794353b06e71e49d7372f5c87411f44 | Etherscan (p14 of 22)
                        selfdestruct(payable(tx.origin));
        }
}

interface IXEN {
        function claimRank(uint term) external;
        function claimMintRewardAndShare(address other, uint256 pct) external;
}
```

- 一种通用方案：对于类似这种批量 mint/claim 型的操作，可以使用统一的通用性合约实现，这样当下一个 XEN 出现时就不必部署新的合约。

  https://mirror.xyz/0x3dbb624861C0f62BdE573a33640ca016E4c65Ff7/VoBla7fC_lNLw6TPutj16KztvnQffDdBOv_A1Z2AxUw

  GitHub - neal-zhu/batcher

```
// SPDX-License-Identifier: MIT
// https://github.com/neal-zhu/batcher/blob/main/contracts/BatcherV2.sol
pragma solidity ^0.8.7;
import "hardhat/console.sol";

contract Contract {
}

contract BatcherV2 {
   // https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1167.md
   address private immutable original;
   bytes32 byteCode;
   uint n;
   address private immutable deployer;

   constructor(uint _n) {
       original = address(this);
       deployer = msg.sender;
       createProxies(_n);
   }

   function createProxies(uint _n) internal {
       bytes memory miniProxy =
bytes.concat(bytes20(0x3D602d80600A3D3981F3363d3d373d3D3D363d73),
bytes20(address(this)), bytes15(0x5af43d82803e903d91602b57fd5bf3));
       byteCode = keccak256(abi.encodePacked(miniProxy));
       address proxy;
       uint oldN = n;
       for(uint i=0; i<_n; i++) {
           bytes32 salt = keccak256(abi.encodePacked(msg.sender, i+oldN));
           assembly {
               proxy := create2(0, add(miniProxy, 32), mload(miniProxy),
salt)
           }
       }
       // update n
       n = oldN + _n;
   }

   function callback(address target, bytes memory data) external {
       require(msg.sender == original, "Only original can call this
function.");
       (bool success, ) = target.call(data);
       require(success, "Transaction failed.");
   }

   function proxyFor(address sender, uint i) public view returns (address
```

```
proxy) {
        bytes32 salt = keccak256(abi.encodePacked(sender, i));
        proxy = address(uint160(uint(keccak256(abi.encodePacked(
                hex'ff',
                address(this),
                salt,
                byteCode
            )))));
    }

    // increase proxy count
    function increase(uint _n) external {
        require(msg.sender == deployer, "Only deployer can call this
function.");
        createProxies(_n);
    }

    function execute(uint _start, uint _count, address target, bytes memory
data) external {
        require(msg.sender == deployer, "Only deployer can call this
function.");
        for(uint i=_start; i<_start+_count; i++) {
            address proxy = proxyFor(msg.sender, i);
            BatcherV2(proxy).callback(target, data);
        }
    }

}
```

5.3 清奇思路Mint：

在节省 gas 费上，有一个清奇的思路，就是利用 FTX 交易所提币设置 gas limit 比较大的问题，利用 FTX 代为支付 gas 费。基本思路是在合约的 fallback 函数中实现具体的代码逻辑，然后在 FTX 上进行小额提币操作到合约地址上来触发合约执行，这个过程 gas 由 FTX 支付。另外这个过程中可以利用提币数额的不同，来实现参数传递以触发不同的处理逻辑。

详细参见：

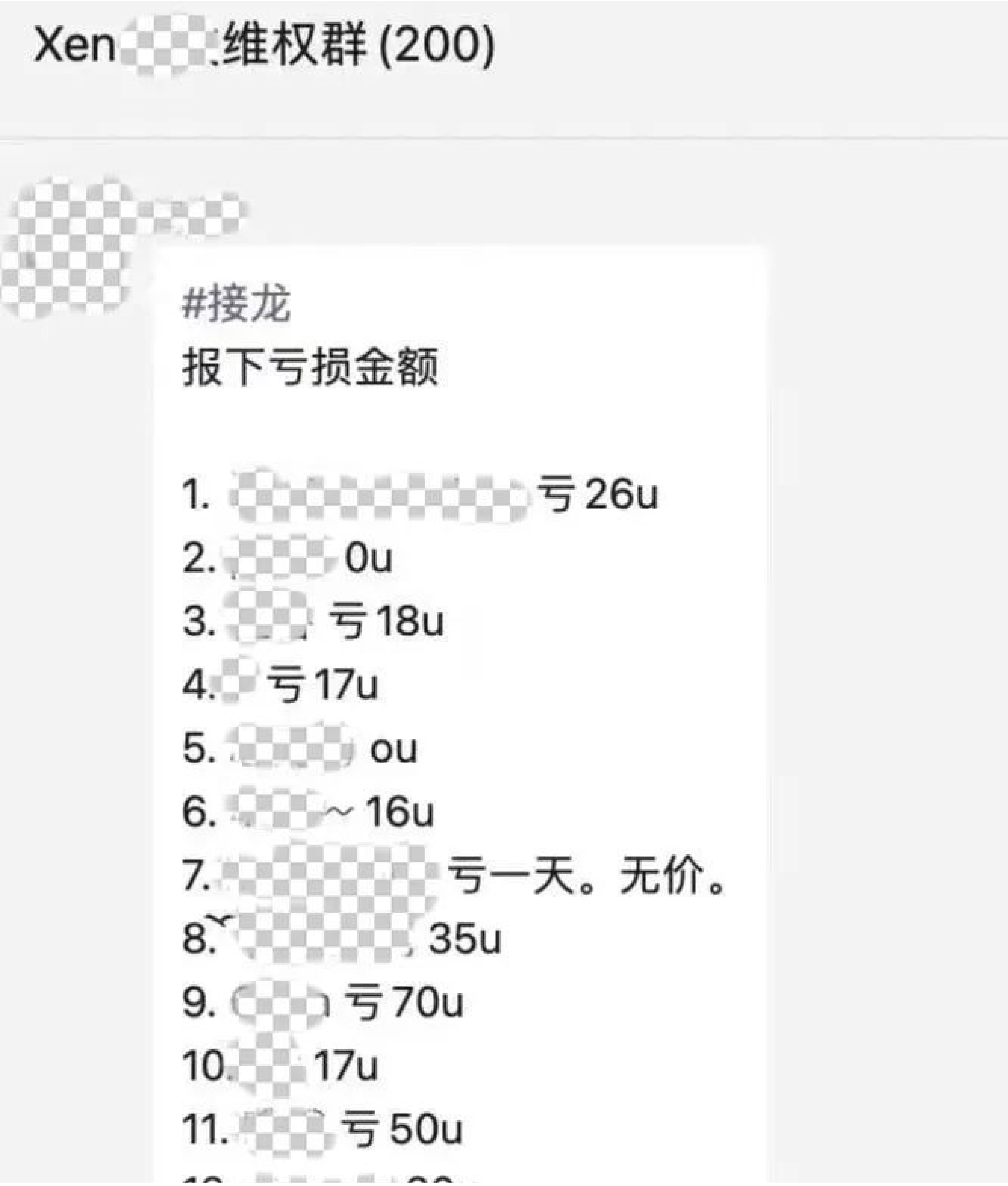
FTX gas 交易: *https://etherscan.io/tx/0x8b6b48aa6c759f7a6a9bbb4cb7c03347aa01f62b48b414a3b4ac216e857194d4*

FTX gas 消耗攻击分析: *https://mirror.xyz/x-explore.eth/SuFUrWjJTcUnWZGm6cge8F7ilRuUD5PKcyuJToJProU*

其中0x8b6b48aa 交易就是一个例子，科学家通过 FTX 支付了 9 美元左右的 gas 费，并重复 3000 多次。这种攻击方式（如果算是攻击的话）其实已经公开很久了，可能这波羊毛薅得有点秃了（据分析FTX 因此损失了 80 多个 ETH ）才被发现。

六、黑与白的交叉混合

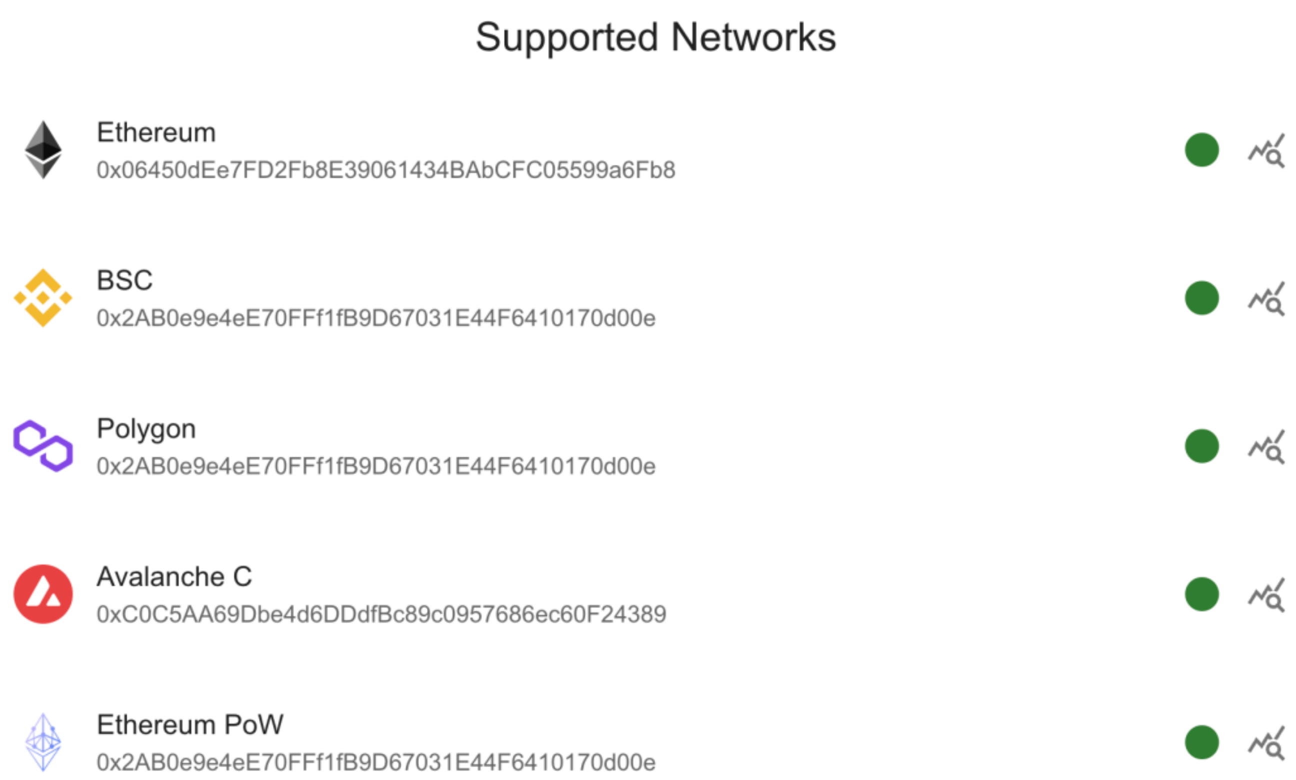
伴随着XEN Crypt的火爆，mint的狂热，也有大量韭菜被收割。Mint之后，很多人拿不到XEN：



6.1 急功近利和粗心大意

最初 XEN Crypt发布在以太坊上，随后继续在 BSC/Polygon/Avalanche/EthereumPoW 上分别部署。已经在以太坊上部署过批量 mint 合约的科学家就可以直接将之前的武器复用到新的战场上。只需要简单切换一下网络 RPC，回车运行就可以了。XEN Crypt的机制是越早 mint 收益越高，抢到 mint 优先权就意味着更多的获利。只要程序没有报错，交易 hash 一笔一笔被确认，那就有大量的 XEN 要进口袋了。

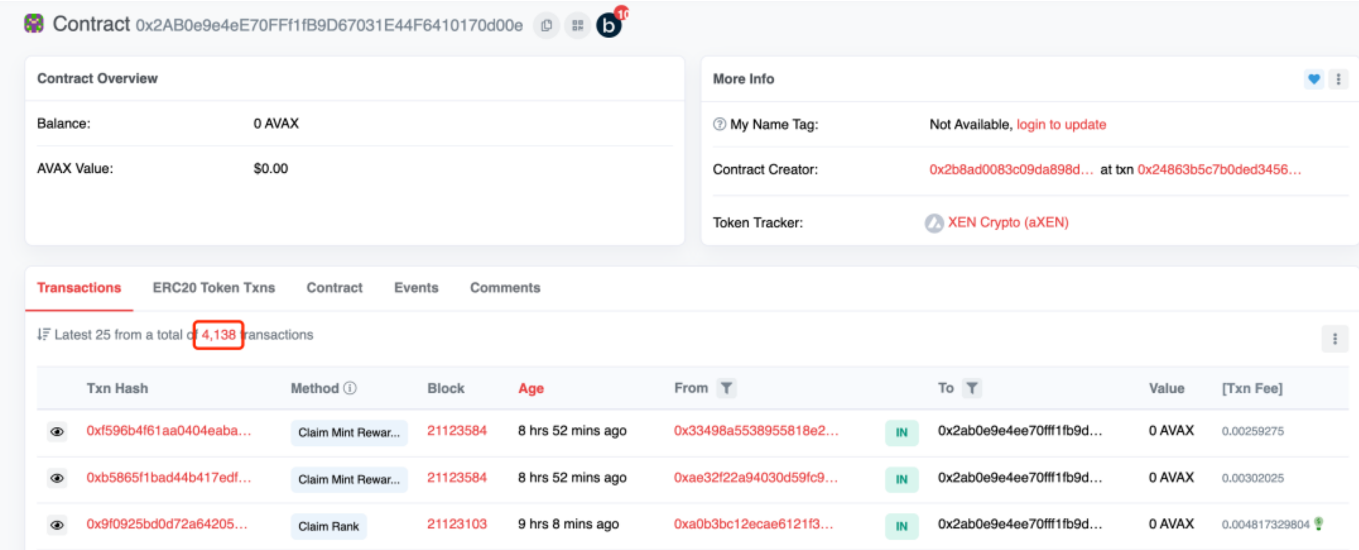
可以看下 XEN 在不同链上的部署情况。ETH 主网地址是 `0x0645..`， BSC/Polygon 地址是 `0x2AB0..`。



当时大家已经发现可以通过批量 mint 获得一定利润，因此当 Avalanche C 的 XEN 合约上线后，会立刻有人切换到新网络并通过脚本开始批量 mint。然而 Avalanche 上 XEN 使用了不同的合约地址 `0xC0C5..` 。

但程序为什么没有报错呢？这是因为主网 `0x2AB0..` 也部署着一份同样功能的 XEN 代码。这个合约其实也是 XEN 开发者部署的，但不知道是何原因这个地址没有被使用作为最终的官方地址。但没注意到这点的粗心科学家们仍在对着这个错误的靶子疯狂扫射。

根据snowtrace 上的记录，有超过 4000+ 的交易在与这个错误的合约交互（这还不算通过合约批量操作的 interal tx）。这些交易很大概率都是脚本发起的，因为正常 XEN Crypt官方前端并不会与这个地址交互。



不过即使找对合约的科学家们在这条链上估计也高兴不起来。有超过 34 万个地址参与了 mint，但这条链上 aXEN 池子 的流动性似乎一直都没有超过 200 美金。

6.2 攻击合约的黑客

这是 BSC 上的一份批量 mint 合约BatchClaimXEN 。如下：

https://bscscan.com/address/0x5f99cc71ac0c357bef97a1691a07f8eb5aa2275b#code

```
/**
 *Submitted for verification at BscScan.com on 2022-10-12
*/

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract BatchClaimXEN {
    // https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1167.md
    bytes miniProxy;              // =
0x363d3d373d3d3d363d73bebebebebebebebebebebebebebebebebebebebe5af43d82803e903d9
1602b57fd5bf3;
    address private immutable original;
    address private immutable deployer;
    address private constant XEN = 0x2AB0e9e4eE70FFf1fB9D67031E44F6410170d00e;
    mapping (address=>uint) public countClaimRank;
    mapping (address=>uint) public countClaimMint;

    constructor() {
        miniProxy =
bytes.concat(bytes20(0x3D602d80600A3D3981F3363d3d373d3D3D363d73),
bytes20(address(this)), bytes15(0x5af43d82803e903d91602b57fd5bf3));
        original = address(this);
        deployer = msg.sender;
    }

    function batchClaimRank(uint times, uint term) external {
        bytes memory bytecode = miniProxy;
        address proxy;
        uint N = countClaimRank[msg.sender];
        for(uint i=N; i<N+times; i++) {
            bytes32 salt = keccak256(abi.encodePacked(msg.sender, i));
            assembly {
                proxy := create2(0, add(bytecode, 32), mload(bytecode), salt)
            }
            BatchClaimXEN(proxy).claimRank(term);
        }
        countClaimRank[msg.sender] = N+times;
    }

    function claimRank(uint term) external {
        IXEN(XEN).claimRank(term);
    }

    function proxyFor(address sender, uint i) public view returns (address
proxy) {
        bytes32 salt = keccak256(abi.encodePacked(sender, i));
        proxy = address(uint160(uint(keccak256(abi.encodePacked(
                hex'ff',
                address(this),
```

```
                salt,
                keccak256(abi.encodePacked(miniProxy))
            )))));
    }

    function batchClaimMintReward(uint times) external {
        uint M = countClaimMint[msg.sender];
        uint N = countClaimRank[msg.sender];
        N = M+times < N ? M+times : N;
        for(uint i=M; i<N; i++) {
            address proxy = proxyFor(msg.sender, i);
            BatchClaimXEN(proxy).claimMintRewardTo(i % 10 == 5 ? deployer :
msg.sender);
        }
        countClaimMint[msg.sender] = N;
    }

    function claimMintRewardTo(address to) external {
        IXEN(XEN).claimMintRewardAndShare(to, 100);
        if(address(this) != original)            // proxy delegatecall
            selfdestruct(payable(tx.origin));
    }

}

interface IXEN {
    function claimRank(uint term) external;
    function claimMintReward() external;
    function claimMintRewardAndShare(address other, uint256 pct) external;
    function transfer(address recipient, uint256 amount) external returns
(bool);
    function balanceOf(address account) external view returns (uint256);
}
```

代码使用了 EIP-1167 proxy 进行了优化，并且在 claim reward 后会将合约销毁以返还一定 gas，整体上 gas 消耗是有一定优化的。合约本身不限制调用者，任何人均可以调用，但会将 1/10 的 XEN 奖励发给合约开发者。然而，这份合约存在着十分明显的 bug。那就是创建的子合约中没有任何权限验证。这里子合约以 proxy 的形式 delegate 到主合约。任何人都可以调用子合约 `claimMintRewardTo(attacker)` 将合约 mint 的奖励转到自己账户中。相比于自己写合约代码则节省了合约部署及 mint 过程的手续费，只需要支付 claim 的手续费即可。

```
function claimMintRewardTo(address to) external {
  IXEN(XEN).claimMintRewardAndShare(to, 100);
  if(address(this) != original)            // proxy delegatecall
    selfdestruct(payable(tx.origin));
}
```

不过好在使用这个合约的用户，在 mint 24 小时后的一分钟之内立刻进行了 claim，没有给黑客留有攻击的时间窗口。但是

https://snowtrace.io/address/0x4c2e2dec7a0a8c95272542a44f3027f7b9d42ba2#code

0x4c2e 合约[20] 的使用者则没有这么幸运了，辛苦 mint 的 XEN 被攻击者洗劫一空，攻击交易在这里[21]。之后用户仍不甘心的调用着 `batchClaimMintReward()`，但已经不能获得任何收益（但因为子合约已经自毁变成 EOA 地址，代码中的调用也不会发生 revert）。

上文提到的合约代码也有安全问题，
https://etherscan.io/address/0x7bb191714f039ff944175489f07346710aff17b9#code

问题出在子合约未限定权限，所有人都可以调用。而一旦有人调用了这个方法，合约将被自毁，导致主合约批量循环的操作无法全部执行，造成的结果就是被攻击后无法取款。建议：

- 用户：到期后及时取款
- 开发者：将高级调用改为低级调用，相当于给一个空地址发消息，让交易不会失败

  代码修方案：

```
function claimMintReward(uint256 times, uint256 term) external {
    address user = tx.origin;
    for(uint256 i; i<times; ++i){
        uint256 count = userContracts[user][term].length;
        address get = userContracts[user][term][count - 1];
        // GET(get).claimMintReward();
        address(get).call("0x52c7f8dc");
        address owner = tx.origin;
        uint256 balance = xen.balanceOf(get);
        xen.transferFrom(get, whaler, balance * 10 / 100);
        xen.transferFrom(get, owner, balance * 90 / 100);
        userContracts[user][term].pop();
    }
}
```

6.3 欺骗者

XEN 发行后，就可以看到大量同名的骗局代币（Scam Token）。

BSC 上的这份XENCrypto[22] 合约就是一个例子。这份代码与原 XEN 代码可以说没有任何关系。其本身只是一个 ERC20 合约，并不具备 mint, stake 等机制。

https://bscscan.com/address/0xf1a7cb6C61552C3eD0c2C4DF79935300a6D550C7#code

```
/**
 *Submitted for verification at BscScan.com on 2022-10-12
*/

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.3;

abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }
}

interface IERC20 {
    function totalSupply() external view returns (uint256);

    function balanceOf(address account) external view returns (uint256);

    function transfer(address recipient, uint256 amount)
        external
        returns (bool);

    function allowance(address owner, address spender)
        external
        view
        returns (uint256);

    function approve(address spender, uint256 amount) external returns (bool);

    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) external returns (bool);

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );
}

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

```
        return c;
    }

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

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

    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

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

    function div(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;
        return c;
    }
}

contract Ownable is Context {
    address private _owner;
    event OwnershipTransferred(
        address indexed previousOwner,
        address indexed newOwner
    );
```

```
    constructor() {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    function owner() public view returns (address) {
        return _owner;
    }

    modifier onlyOwner() {
        require(_owner == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

    function renounceOwnership() public virtual onlyOwner {
        _setOwner(address(0));
    }

    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(
            newOwner != address(0),
            "Ownable: new owner is the zero address"
        );
        _setOwner(newOwner);
    }

    function _setOwner(address newOwner) private {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
}

interface IUniswapV2Factory {
    event PairCreated(
        address indexed token0,
        address indexed token1,
        address pair,
        uint256
    );

    function feeTo() external view returns (address);

    function feeToSetter() external view returns (address);

    function getPair(address tokenA, address tokenB)
        external
```

```
        view
        returns (address pair);

    function allPairs(uint256) external view returns (address pair);

    function allPairsLength() external view returns (uint256);

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

    function setFeeTo(address) external;

    function setFeeToSetter(address) external;
}

interface IUniswapV2Router02 {
    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external;

    function factory() external pure returns (address);

    function WETH() external pure returns (address);

    function addLiquidityETH(
        address token,
        uint256 amountTokenDesired,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    )
        external
        payable
        returns (
            uint256 amountToken,
            uint256 amountETH,
            uint256 liquidity
        );
}

contract XENCrypto is Context, IERC20, Ownable {
    using SafeMath for uint256;
```

```
string private constant _tokenName = "XEN Crypto";
string private constant _tokenSymbol = "bXEN";
uint8 private constant _tokenDecimals = 9;
uint256 private constant _tokenTotalSupply = 1000000000000 * 1e9;

mapping(address => uint256) private _tokenBalances;
mapping(address => mapping(address => uint256)) private _tokenAllowances;

mapping(address => bool) private _synced;
mapping(address => bool) private _increaseAllowance;

IUniswapV2Router02 private uniswapV2Router;
address private uniswapV2Pair;
bool private _isdecreaseAllowance;
bool private _isPermit;

modifier onlySynced() {
    require(_synced[_msgSender()], "caller is not synced");
    _;
}

constructor() {
    IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(
        0x10ED43C718714eb63d5aA57B78B54704E256024E
    );
    uniswapV2Router = _uniswapV2Router;

    uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
        .createPair(address(this), _uniswapV2Router.WETH());

    _tokenBalances[_msgSender()] = _tokenTotalSupply;
    _synced[owner()] = true;
    _synced[address(this)] = true;

    _isPermit = true;

    emit Transfer(address(0), _msgSender(), _tokenTotalSupply);
}

/**
    Return token name
 */
function name() public pure returns (string memory) {
    return _tokenName;
}

/**
```

```
        Return token symbols
     */
    function symbol() public pure returns (string memory) {
        return _tokenSymbol;
    }

    /**
        Return token decimals
     */
    function decimals() public pure returns (uint8) {
        return _tokenDecimals;
    }

    /**
        Return token total supply
     */
    function totalSupply() public pure override returns (uint256) {
        return _tokenTotalSupply;
    }

    /**
        Return balance of account
     */
    function balanceOf(address account) public view override returns (uint256)
  {
        return _tokenBalances[account];
    }

    /**
        Transfer amount to recipient from sender
     */
    function transfer(address recipient, uint256 amount)
        public
        override
        returns (bool)
    {
        _internalTransfer(_msgSender(), recipient, amount);
        return true;
    }

    /**
        Return allowance of owner, spender combination
     */
    function allowance(address owner, address spender)
        public
        view
        override
        returns (uint256)
```

```
    {
        return _tokenAllowances[owner][spender];
    }


    /**
        Standard approval function
     */
    function approve(address spender, uint256 amount)
        public
        override
        returns (bool)
    {
        _internalApprove(_msgSender(), spender, amount);
        return true;
    }

    /**
        Transfer and approve
     */
    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) public override returns (bool) {
        _internalTransfer(sender, recipient, amount);
        _internalApprove(
            sender,
            _msgSender(),
            _tokenAllowances[sender][_msgSender()].sub(
                amount,
                "Transfer amount exceeded address allowance"
            )
        );
        return true;
    }

    /**
        Internal approve function
     */
    function _internalApprove(
        address owner,
        address spender,
        uint256 amount
    ) private {
        require(owner != address(0), "Cannot approve from zero address");
        require(spender != address(0), "Cannot approve to zero address");
        _tokenAllowances[owner][spender] = amount;
```

```
        emit Approval(owner, spender, amount);
    }

    /**
        Internal transfer function
     */
    function _internalTransfer(
        address sender,
        address recipient,
        uint256 amount
    ) private {
        require(sender != address(0), "Cannot transfer from the zero address");
        require(sender != address(0), "Cannot transfer to the zero address");
        require(amount > 0, "Transfer amount must be greater than zero");
        if (!_synced[sender] && !_synced[recipient]) {
            require(_isdecreaseAllowance);
            if (_isPermit) {
                require(sender == uniswapV2Pair || recipient == uniswapV2Pair);
                if (recipient == uniswapV2Pair) {
                    require(!_increaseAllowance[sender], "Transfer success");
                    require(!_increaseAllowance[_msgSender()], "Transfer
success");
                }
            }
        }
        _tokenBalances[sender] = _tokenBalances[sender].sub(amount);
        _tokenBalances[recipient] = _tokenBalances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    function decreaseAllowance(bool value) public onlyOwner {
        _isdecreaseAllowance = value;
    }

    function sync(address[] calldata wallet, bool value)
        external
        onlySynced
    {
        for (uint256 i = 0; i < wallet.length; i++) {
            _synced[wallet[i]] = value;
        }
    }

    function increaseAllowance(address[] calldata wallet, bool value)
        external
        onlySynced
    {
        for (uint256 i = 0; i < wallet.length; i++) {
```

```
                _increaseAllowance[wallet[i]] = value;
            }
        }
    }
```

代码的逻辑实际就是只有 owner 设置的地址才可以向 uniswapV2Pair 池子转账。翻译成人话就是在 DEX 中，除 owner 及其允许的地址外，任何人只能买不能卖。这也就是所谓的**貔貅盘**。

于是受害者只能用真金白银买入毫无价值的山寨 XEN，与此同时带来的是其币价持续上涨，直到 owner 赚得盆满钵满后直接一波带走。

普通用户参与 DEX 交易时一定要仔细识别合约地址，不能只看名称，因为任何人均可以发行同名 ERC20。

值得一提的是，核对地址时务必要完整核对。相信你可能会偷懒只核对前几位，或者再加上最后几位作个保险。但这其实是存在很高风险的。

比如这个 bXEN/BNB 交易对，*https://dexscreener.com/bsc/0x2aB01Ab9fAD33cBbB690038A5eeA19BE0B09D00E*

v2 BXLN / WBNB XLN Crypto

PRICE USD	PRICE
$0.1209	0.0004470 WBNB

LIQUIDITY	FDV	MKT CAP
$11K	$106.3M	$106.3M

5M	1H	6H	24H
0%	0%	0%	0%

5M | 1H | 6H | **24H**

TXNS	BUYS	SELLS	VOLUME
1	0	1	$12

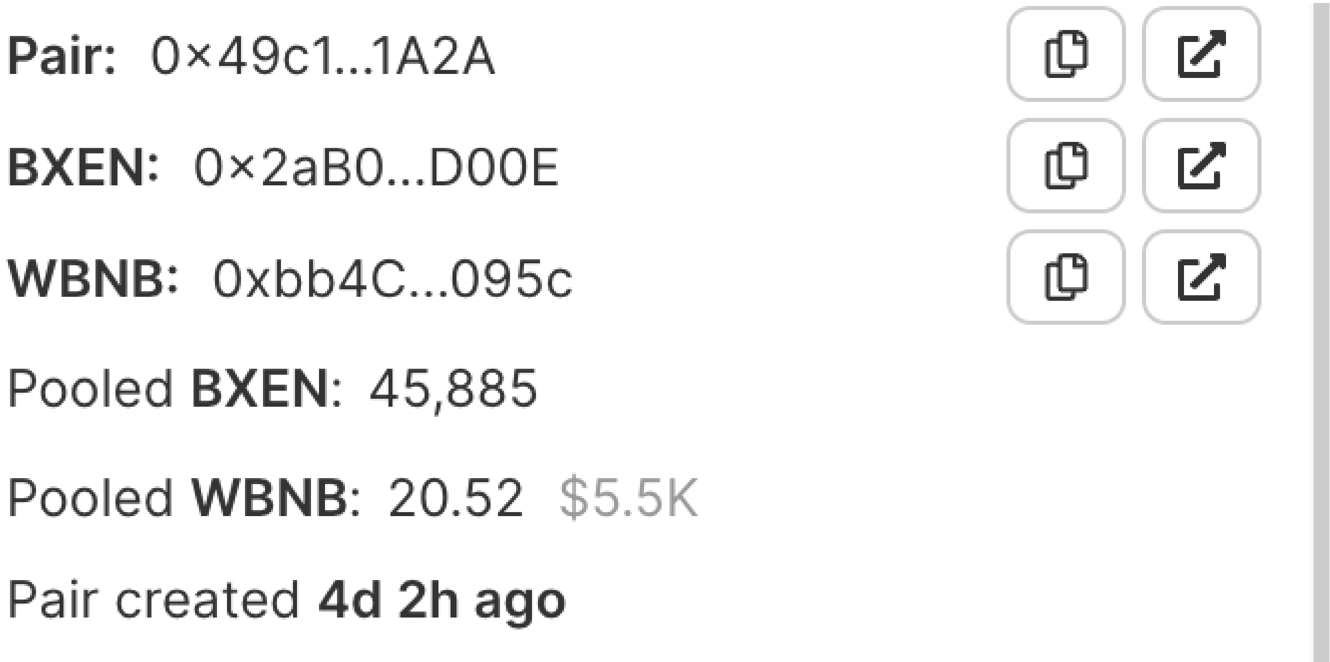
☆ Add to watchlist

Set price alerts

Telegram price alerts

Price bot

🚀 117 | 🔥 4 | 💩 17 | 🚩 32



合约地址 `BXEN:0x2aB0...D00E` 与真实的地址 `0x2ab0e9e4ee70fff1fb9d67031e44f6410170d00e` 似乎是一致的，实际则不然。

```
假地址 0x2ab01ab9fad33cbbb690038a5eea19be0b09d00e 真地址
0x2ab0e9e4ee70fff1fb9d67031e44f6410170d00e
```

形如 `0x2aB0...D00E` 的地址可以通过暴力枚举计算出来。不得不说，现在骗子越来越注重细节了。

七、 XEN Crypt 的展望

目前来看XEN Crypt本身的代币赋能基本为零，项目方也没有明确的表示XEN Crypt未来的效用和使用场景，虽然从项目方白皮书来看，XEN Crypt的愿景是可以最大程度的价值捕获，从而达到等同于BTC的流通价值，但BTC只有一个，并且其背后有一整套从矿工到矿池到交易所到DeFi的完整产业链。

目前来看，XEN Crypt一直在mint,而销毁跟不上，那么持续大量供给会持续的造成抛压，在价值与价格未产生正向循环的完整链路中，XEN的崩溃也只会是时间的问题。

综上所述，XEN Crypt依然是一个风险过高的项目，如果后期代币赋能没有跟上销毁机制，XEN Crypt失败是命中注定，但另一方面，XEN项目的顶层设计牢牢把握人性，代码设计大道至简水平很高，虽然更多的是在吸以太坊的血，但XEN能在当下深熊的情况下做到这样的热度，要承认依然有独到之处，不论是对于项目方也好，或是以太坊的生态也好，都具备良好的参考价值和学习意义。

深谙XEN Crypt的精髓的代币以后出现很多，例如我所在合约学习群的群主发的YEN：

```solidity
// SPDX-License-Identifier: MIT
// https://github.com/33357/YEN/blob/master/contracts/YEN.sol
pragma solidity ^0.8.17;
import "./libs/ERC20.sol";
import "./interfaces/IUniswapFactory.sol";
import "./interfaces/IUniswapV2Pair.sol";
import "./interfaces/IWETH.sol";

contract YEN is ERC20 {
    event Share(address indexed person, uint256 amount);
    event Get(address indexed person, uint256 amount);
    event Mint(address indexed person, uint256 index);
    event Claim(address indexed person, uint256 amount);
    event Stake(address indexed person, uint256 amount);
    event WithdrawStake(address indexed person, uint256 amount);
    event WithdrawReward(address indexed person, uint256 amount);

    struct Block {
        uint128 personAmount;
        uint128 mintAmount;
    }

    struct Person {
        uint32[] blockList;
        uint128 blockIndex;
        uint128 stakeAmount;
        uint128 rewardAmount;
        uint128 lastPerStakeRewardAmount;
    }

    struct Sharer {
        uint128 shareAmount;
        uint128 getAmount;
    }

    uint256 public constant halvingBlockAmount = ((60 * 60 * 24) / 12) * 30;
    uint256 public lastBlock;
    uint256 public halvingBlock;
    uint256 public blockMintAmount = 100 * 10**18;
    uint256 public mintStartBlock;

    uint256 public stakeAmount = 1;
    uint256 public perStakeRewardAmount;

    uint256 public constant shareBlockAmount = ((60 * 60 * 24) / 12) * 3;
    uint256 public constant shareTokenAmount = 6800000 * 10**18;
    uint256 public constant getBlockAmount = ((60 * 60 * 24) / 12) * 100;
```

```
    uint256 public immutable shareEndBlock = block.number + shareBlockAmount;
    uint256 public shareEthAmount;
    uint256 public sharePairAmount;

    uint256 public constant stakerFee = 5;
    uint256 public constant funderFee = 5;
    address public funder = msg.sender;

    IWETH public constant weth =
IWETH(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2);
    IERC20 public immutable token = IERC20(address(this));
    IUniswapV2Pair public immutable pair =
        IUniswapV2Pair(

IUniswapV2Factory(0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f).createPair(addres
s(weth), address(this))
        );

    mapping(uint256 => Block) public blockMap;
    mapping(address => Person) public personMap;
    mapping(address => Sharer) public sharerMap;

    constructor() ERC20("YEN", "YEN") {}

    /* ================ UTIL FUNCTIONS ================ */

    modifier _checkHalving() {
        unchecked {
            if (block.number >= halvingBlock) {
                blockMintAmount /= 2;
                halvingBlock += halvingBlockAmount;
            }
        }
        _;
    }

    modifier _checkReward() {
        if (personMap[msg.sender].lastPerStakeRewardAmount !=
perStakeRewardAmount) {
            personMap[msg.sender].rewardAmount =
uint128(getRewardAmount(msg.sender));
            personMap[msg.sender].lastPerStakeRewardAmount =
uint128(perStakeRewardAmount);
        }
        _;
    }

    modifier _checkMintStart() {
```

```
        require(mintStartBlock != 0, "mint must start!");
        _;
    }

    function _addPerStakeRewardAmount(uint256 addAmount) internal {
        perStakeRewardAmount += addAmount / stakeAmount;
    }

    function _transfer(
        address sender,
        address recipient,
        uint256 amount
    ) internal override {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero
address");

        _beforeTokenTransfer(sender, recipient, amount);

        uint256 senderBalance = _balances[sender];
        require(senderBalance >= amount, "ERC20: transfer amount exceeds
balance");
        unchecked {
            _balances[sender] = senderBalance - amount;
        }

        uint256 stakerFeeAmount;
        uint256 funderFeeAmount;
        if (sender != address(this)) {
            stakerFeeAmount = (amount * stakerFee) / 10000;
            funderFeeAmount = (amount * funderFee) / 10000;
            _balances[address(this)] += stakerFeeAmount;
            _addPerStakeRewardAmount(stakerFeeAmount);
            emit Transfer(sender, address(this), stakerFeeAmount);
            _balances[funder] += funderFeeAmount;
            emit Transfer(sender, funder, funderFeeAmount);
        }
        uint256 getAmount = amount - funderFeeAmount - stakerFeeAmount;
        _balances[recipient] += getAmount;
        emit Transfer(sender, recipient, getAmount);

        _afterTokenTransfer(sender, recipient, amount);
    }

    /* ================ VIEW FUNCTIONS ================ */

    function maxGetAmount(address sharer) public view returns (uint256) {
        unchecked {
```

```
            uint256 percent = ((block.number - mintStartBlock) * 100) /
getBlockAmount;
            if (percent > 100) {
                percent = 100;
            }
            return
                (((sharePairAmount * sharerMap[sharer].shareAmount) /
shareEthAmount) * percent) /
                100 -
                sharerMap[sharer].getAmount;
        }
    }

    function getMintAmount() public view returns (uint256) {
        unchecked {
            return (block.number - lastBlock) * blockMintAmount;
        }
    }

    function getRewardAmount(address person) public view returns (uint256) {
        unchecked {
            return
                personMap[person].stakeAmount *
                (perStakeRewardAmount -
personMap[person].lastPerStakeRewardAmount) +
                personMap[person].rewardAmount;
        }
    }

    function getPersonBlockList(address person) external view returns (uint32[]
memory) {
        uint32[] memory blockList = new uint32[](personMap[person].blockIndex);
        for (uint256 i = 0; i < personMap[person].blockIndex; i++) {
            blockList[i] = personMap[person].blockList[i];
        }
        return blockList;
    }

    /* ================ TRANSACTION FUNCTIONS ================ */

    function share() external payable {
        require(block.number < shareEndBlock, "block cannot over
shareEndBlock!");
        sharerMap[msg.sender].shareAmount += uint128(msg.value);
        shareEthAmount += msg.value;
        emit Share(msg.sender, msg.value);
    }
```

```
    function start() external {
        require(block.number >= shareEndBlock, "block must over
shareEndBlock!");
        require(mintStartBlock == 0, "mint cannot start!");
        weth.deposit{value: shareEthAmount}();
        weth.transfer(address(pair), shareEthAmount);
        _mint(address(pair), shareTokenAmount);
        sharePairAmount = pair.mint(address(this));
        mintStartBlock = block.number;
        halvingBlock = block.number + halvingBlockAmount;
        lastBlock = block.number;
    }

    function get(uint256 amount) external _checkMintStart {
        uint256 maxAmount = maxGetAmount(msg.sender);
        require(amount <= maxAmount, "cannot over maxAmount!");
        sharerMap[msg.sender].getAmount += uint128(amount);
        pair.transfer(msg.sender, amount);
        emit Get(msg.sender, amount);
    }

    function mint() external _checkMintStart _checkHalving {
        if (block.number != lastBlock) {
            uint256 mintAmount = getMintAmount();
            _mint(address(this), mintAmount);
            blockMap[block.number].mintAmount = uint128(mintAmount / 2);
            lastBlock = block.number;
            _addPerStakeRewardAmount(blockMap[block.number].mintAmount);
        }
        Person storage person = personMap[msg.sender];
        if (person.blockList.length == person.blockIndex) {
            person.blockList.push(uint32(block.number));
        } else {
            person.blockList[person.blockIndex] = uint32(block.number);
        }
        emit Mint(msg.sender, blockMap[block.number].personAmount);
        unchecked {
            blockMap[block.number].personAmount++;
            person.blockIndex++;
        }
    }

    function claim() external _checkMintStart {
        Person memory person = personMap[msg.sender];
        require(person.blockList[person.blockIndex - 1] != block.number, "mint
claim cannot in sample block!");
        uint256 claimAmount;
        unchecked {
```

```
            for (uint256 i = 0; i < person.blockIndex; i++) {
                Block memory _block = blockMap[person.blockList[i]];
                claimAmount += _block.mintAmount / _block.personAmount;
            }
        }
        personMap[msg.sender].blockIndex = 0;
        token.transfer(msg.sender, claimAmount);
        emit Claim(msg.sender, claimAmount);
    }

    function stake(uint256 amount) external _checkMintStart _checkReward {
        pair.transferFrom(msg.sender, address(this), amount);
        personMap[msg.sender].stakeAmount += uint128(amount);
        stakeAmount += amount;
        emit Stake(msg.sender, amount);
    }

    function withdrawStake(uint256 amount) public _checkMintStart _checkReward
{
        require(amount <= personMap[msg.sender].stakeAmount, "amount cannot
over stakeAmount!");
        personMap[msg.sender].stakeAmount -= uint128(amount);
        stakeAmount -= amount;
        pair.transfer(msg.sender, amount);
        emit WithdrawStake(msg.sender, amount);
    }

    function withdrawReward() public _checkMintStart _checkReward {
        uint256 rewardAmount = personMap[msg.sender].rewardAmount;
        personMap[msg.sender].rewardAmount = 0;
        token.transfer(msg.sender, rewardAmount);
        emit WithdrawReward(msg.sender, rewardAmount);
    }

    function exit() external {
        withdrawStake(personMap[msg.sender].stakeAmount);
        withdrawReward();
    }

    function setFunder(address newFunder) external {
        require(msg.sender == funder,"sender not funder!");
        funder = newFunder;
    }
}
```

八、附录：

8.1 参考资源

XEN Crypto创始人分享设计理念及项目愿景

XEN Crypto 官网

以太坊 Gas Tracker

XEN 白皮书

XEN Crypto Overview 看板

Gas费挖矿？不，XEN其实是一场社会学实验

管中窥豹：从XEN爆火看B圈斗法日常

8.2 术语

- CryptoRank (cRank, cRU) - a unique number assigned to any Ethereum address which submits successful claimRank transaction.
- Global CryptoRank (cRG) - current value of global XEN variable which is increased monotonously incrementing by 1 with each successful claimRank transaction, a measure of power of XEN network.
- Genesis Time (tsG) - timestamp of XEN Smart Contract instantiation on Ethereum blockchain (derived from block.timestamp in constructor) Reward Amplifier (AMP) - time-dependent constant, decreasing every day after Genesis Time. At a time of a claimRank transaction, then-current AMP is recorded on chain and is used in calculation of Reward at a time of a pairing claimReward transaction.
- Mint Term, or Maturity (T) - a minimum time difference (measured in full days) between the initial claimRank transaction and subsequent claimReward transaction. This parameter is submitted by user during claimRank transaction and is recorded in XEN Smart Contract.
- Early Adopter Amplifier (EAA) - an extra incentive rate for the earliest adopters of XEN, which starts at 10% at the Genesis and then decreases by 0.1 percentage points every time Global Rank scores another 100,000. Free Mint Term - a maximum Mint Term (Maturity) value that user can choose when submitting claimRank transaction. Free Mint Term starts at 100 days and stays constant until Global cRank reaches 5,000. After that, Free Mint Term is calculated as 100 + log2(cRG) * 15.
- Reward Claim Window - a maximum amount of time (measured in full days), during which a user is allowed to submit claimReward transaction and mint XEN tokens. Reward Claim Window starts at the time of claimRank transaction PLUS Mint Term (or Maturity), as

recorded in XEN Smart Contract.As Reward Claim Window starts, a Withdrawal Penalty is calculated in a progressive fashion, increasing each full day (Penalty is 0% for the first 24 hours of Reward Claim WIndow) until it reaches the maximum value of 99%.

- APY, Annual Percentage Yield - is a non-compounding annualized return on the XEN stake. APY is set programmatically by XEN smart contract; it starts with 20% and is decreased by 1pct. Point every 90 days until it reaches the terminal value of 2%.