THE TAO

道法自然

MAR 23, 2021

V1.1

https://thetao.cash

目录

前言

斗转星移，时光荏苒，从比特币的诞生到现在已经走过 10 个年头有余，整个加密货币的世界随着比特币价格的起伏，时而高潮，时而低谷。期间不乏“后起之秀”，或如以太坊般独领风骚，或如 EOS 般一地鸡毛，但岁月悠悠，虽尚无十全十美之币，也偶尔出现小清新让世人眼前一亮，就像 BHD 以及 Grin。本文拟就近 10 年币圈之美，阐述当下加密货币世界格局和不足，并试着提出一个新的矿币，基于 zk-SNARKs 的硬盘容量证明（PoC）矿币，The TAO（道）。

道，是天地万物的演化运行机制，中国哲学的信念之一。老子认为道决定了事物“有”或“无”、以及生物“生”或“死”的存在形式；从无到有、从有到无和周而复始的自然现象，是万事万物在道协同作用下所产生的结果；“人法地，地法天，天法道，道法自然”的理念是「自然而然」，“道”虽是生长万物的，却是无目的、无意识的，它“生而不有，为而不恃，长而不宰”，即不把万物据为己有，不夸耀自己的功劳，不主宰和支配万物，而是听任万物自然而然发展着。

-- 维基百科（https://zh.wikipedia.org/wiki/道）

背景

币圈天下大势，分三大主流共识，分别为工作量证明（PoW），股权证明（PoS）和容量证明（PoC）。其他都是在此三类之上的演化，而 DAG 根本不是区块链，不在本文讨论范围内。至于其他闭门造车的共识，除了哗众取宠之外，在数学上根本站不住脚，不值一提。

隐私币则从 Utility Token（功能代币）的角度，证明了自己的存在价值。

三大共识

PoW 是最早，也是最成熟的共识算法，代表作有比特币及其分叉币，以太坊及其分叉币，以及莱特币，Zcash，门罗等等，其主要特点是矿工用链规定的各类加密哈希算法在规定时间内暴力穷举出满足共识规则的 nonce 值，从而获得记账权，并得到打包区块的奖励。PoW 常因为巨大的电力为世人所诟病，而日益增多的哈希算法专用芯片（ASIC）化，让 PoW 矿币成为重资产项目，成为中小散户望而却步、只有资本大鳄才玩得起的挖矿币种。

股权证明，PoS，历史不短，但价格却一直平淡，究其原因，不外乎如下几条。首先，所有币在“创世块”产生并完成初始分配，不再需要通过挖矿产生；其次，股权（stake）的过度集中容易产生贿选，Steemit 的社区分化，就是最新的例子，这不是第一次，也不会是最后一次，只是比较有名的一次；第三，PoS 的币通常总量巨大，且考虑到出块节点的激励，一般采用通胀模型；第四，迄今为止，PoS 并未找到很好的实用功能，即便以太坊 2.0 即将采用 PoS，社区暂时依然看不到光明的未来；第五，Staking 并不能救 PoS。但 PoS 并不是一无是处，PoS 更适合做一个类似公司化的社区治理的 stake 而不是作为 utility token。

PoC，容量证明，从诞生之日起就注定与众不同。PoC 大致分两大方向，第一类是以 Burst 为代表的，它采用了时空转换的原理，将 PoW 里的加密哈希预先集中计算并写入硬盘，挖矿时，矿工程序只需扫盘得到 nonce，通过计算提交一个自己最小 deadline，每轮全网最小 deadline 者获得记账权，打包出块同时获得激励。此类 PoC 入门门槛低，普通电脑只要有硬盘即可参与挖矿，且电费消耗极低，却在一定的算力情况下，达到与 PoW 一样的安全特性。Burst 作为第一个完整的、能运行的 PoC 共识公链，证明了 PoC 的可行性，只是后期运营欠佳，成了先烈。BHD 作为 PoC 的后起之秀，在大量哺育市场的同时，证明了自己的价值；另一类是 Filecoin 为代表的 PoC+PoW 混合共识，虽然大类也是 PoC 共识，但其要求硬盘容量匹配

足够大的 PoW 算力（显卡或者 ASIC），使得整个共识算法变得超级复杂，给矿工带来了极高技术门槛的同时，初投入资产门槛高到比肩纯 PoW 共识。

隐私币

说到隐私币，大家会立即想到 Zcash，门罗和 Grin。

Zcash 当年从比特币代码树分叉出来，改掉了共识的哈希算法，并在业务逻辑上加入了零知识证明（zk-SNARKs），曾经的名噪一时，成为隐私币的龙头老大，只是好景不长，时下已经到了算力下降到比较容易被 51%攻击的地步。

门罗走的是环签名的隐私算法路子，几年来跌跌撞撞，不断在跟 ASIC 对抗中，修修补补，也是僵尸挖矿网络的首选币种，一直处于毁誉参半之中。

Grin 是 2019 年的隐私币新星，基本原理是给比特币代码树加入了加法同态来来实现隐私性，但正因为此，其隐私只能建立在所有人都在线的前提下，可以认为是一种在线混币链。必须在线，给实际使用带来了很多麻烦和困扰。

以上隐私币有一个共同点，那就是都是 PoW 共识，都需要巨大的电力耗费来支撑网络安全。

但隐私币却一直是加密货币世界的“强刚需”！作为 Utility Token，隐私本应该是加密货币天然的、与生俱来的属性，可惜的是，比特币和以太坊这类主流币种并不具备隐私性，使得社区对隐私币的探索从未停止过。

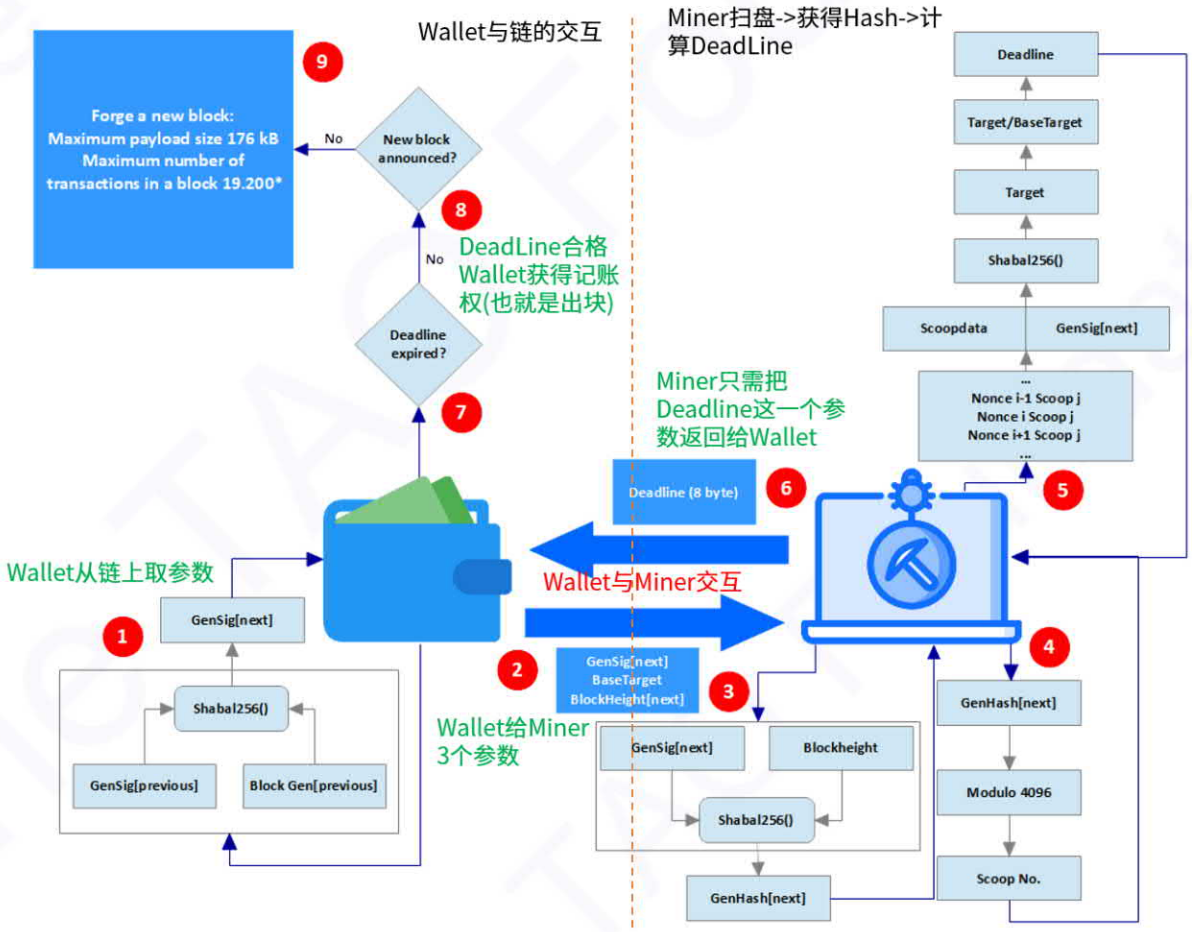
THE TAO

容量证明（PoC）是优秀的共识算法，参与门槛低，耗电量少，无法 ASIC 化，公开透明。而零知识证明（zk-SNARKs）经历了时间的洗礼，被证实是一种可靠的隐私交易算法。The TAO 正是结合二者所长而成，底层挖矿为 PoC 共识算法，上层业务逻辑为 zk-SNARKs。虽然坊间不乏各种号称做到 PoC + zk-SNARKs 的公链，但稍加考证即可发现，根本就是伪隐私币。The TAO 是世界第一个 PoC + zk-SNARKs 的隐私币！

技术架构

一. 挖矿过程

挖矿是通过提交满足链规定的最小 deadline 从而获得记账权并获得 token 激励的过程。The TAO 的挖矿是矿工定时扫盘，获得nonce，计算deadline并提交给钱包或者矿池来完成的。分solo（独立挖矿）和pool（矿池挖矿）两类，矿工可根据自己需要自由选择用何种方式。

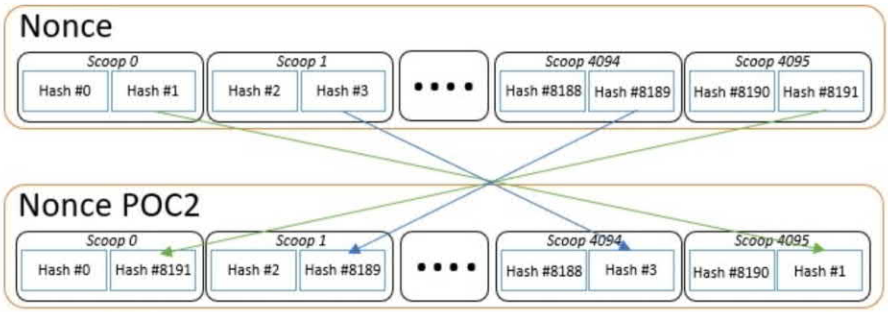


如图所示，挖矿分 9 个步骤：1）钱包与链交互，获得当下链参数；2）钱包将链参数给到矿工程序；3）矿工程序根据当前链参数计算 nexthash 值；4）矿工程序根据 nexthash 值计算当前 Scoop 号；5）矿工把自己硬盘容量下当前 Scoop 号中所有 nonce 扫到并计算 deadline 值；6）把 deadline 值提交给钱包；7）钱包做本地判断，收到的 deadline 值是否大于自己设定的默认最大值，如果大于这个值，则直接丢弃，并不会再提交到链上；8）钱包把 deadline 值提交到链上；9）链做判断，接受的 deadline 是否为当前所有中最小，如果最小，给予记账权和 token 奖励，否则，丢弃次 deadline 值。

挖矿的过程中，钱包可以跟矿工程序在一台物理机器上，也就是 solo 模式，也可以在远端，即 pool 模式。The TAO 官方会跟社区共同运营一个 pool，但并不要求所有矿工都必须参加官方 pool，当然鼓励大家参与官方 pool。矿工程序有 C++及 Rust 两类，分别可以在 Windows 以及 Linux 下运行，miner 程序将在官网获取，且强烈建议社区只从官网下载相关程序。

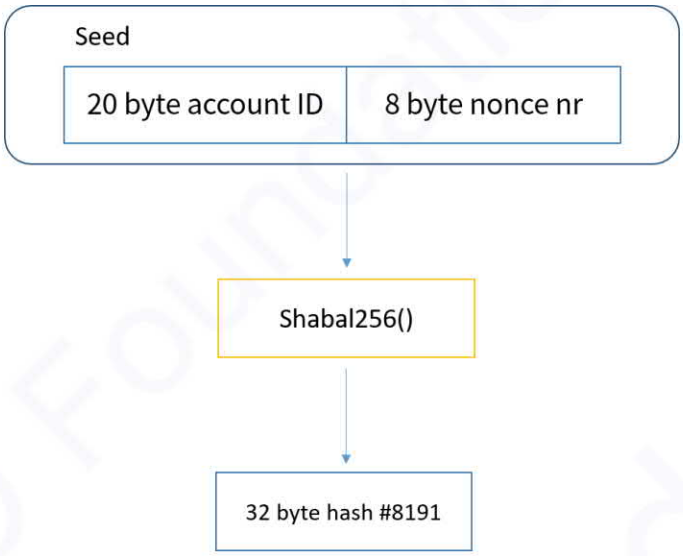
二．P 盘过程

The TAO 采用了 PoC 的共识，也就会在挖矿之前，需要执行一个制作 plot 的过程，此过程称为 P 盘。The TAO 的 plot 文件采用 PoC2 格式，与 PoC 的格式区别如下：

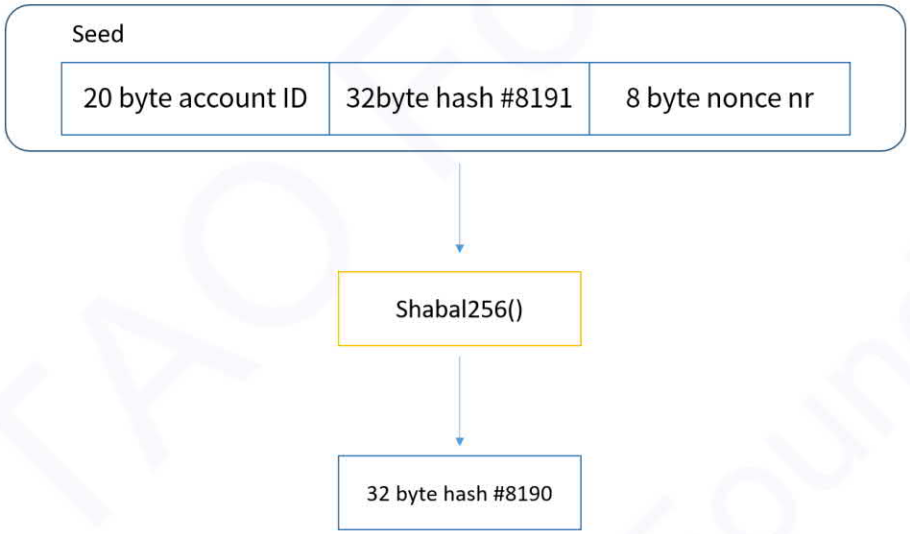


为减少硬盘磁头频繁寻址，PoC2 的格式做了优化，将同一 Scoop 号的 nonce 放在了一起，从而提高了效率，延长了硬盘使用寿命。

The TAO 的 nonce 计算与 Burst、BHD 等略有不同，具体体现在 seed 的生成上。The TAO 用了 20 个字节的账户 ID 而不是通常的 8 字节的，且并不是简单从 8 字节延长到 20 个字节，具体算法将会在开源的 p 盘程序中体现。加密哈希算法采用 Shabal256，The TAO 的 nonce 计算过程如下：



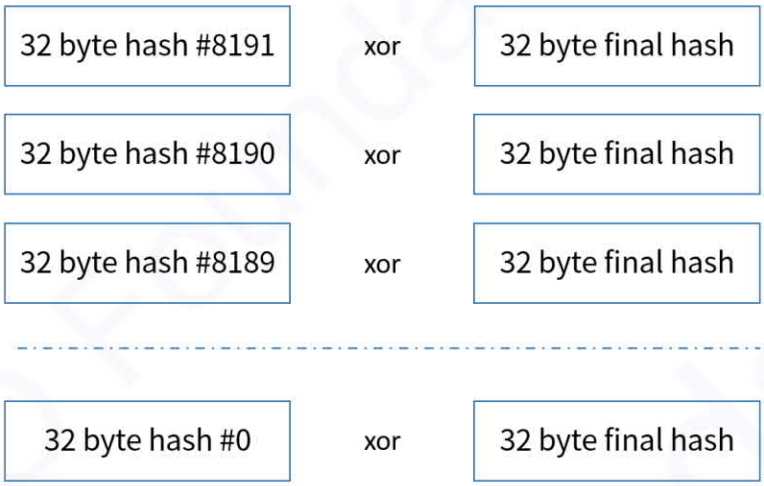
通过种子，用 Shabal256 算法生成第一个哈希，也就是号码为 8191 的哈希，然后将结果作为种子的一部分，用同样的算法生成第二个哈希，#8190：



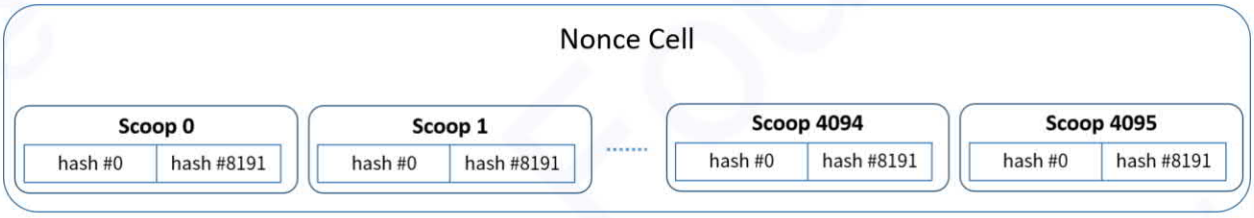
依次递归，直到生成所有 8192 个哈希。此时需要生成一个叫做 final hash 的哈希，其种子为所有 8192 个哈希加上最早的 28 个字节的原始种子，即

Final hash = Shabal256（Hash #0~8191 + startedseed(20byte account ID+8byte hash nr））

最后，final hash 将于之前生成的每一个哈希做 xor 计算，从而得到最终的 nonce：



将 8192 个哈希结果按相邻两个为一组排列，每一组称为一个 Scoop，得到 4096 个 Scoop，填入 Cell 中。由此，Cell 构造完成。



在生成 Cell 的过程中，计算机必须利用缓存记录所有的中间结果，才能获取最终的哈希结果。此时可以用 SSD 辅助，以提高 p 盘效率。

由于每个 Cell 包含 8192 个 Shabal256 哈希结果，每个哈希结果长度 32byte，因此每个 Cell 将固定占用 256KB 空间。

反复进行生成 Cell 的操作，再将所有 Cell 进行优化排列，填满 Plot 文件。

The TAO 官方将免费向社区提供基于 Rust 语言的带显卡加速的 p 盘程序。

基于 Rust 的核心挖矿代码片段如下：

```rust
use crate::miner::{Buffer, NonceData};
use crate::ocl::GpuContext;
use crate::ocl::{gpu_hash, gpu_transfer};
use crate::reader::ReadReply;
use crossbeam_channel::{Receiver, Sender};
use futures::sync::mpsc;
use futures::{Future, Sink};
use std::sync::Arc;
use std::u64;
use hex;

pub fn create_gpu_worker_task(
    benchmark: bool,
    rx_read_replies: Receiver<ReadReply>,
    tx_empty_buffers: Sender<Box<dyn Buffer + Send>>,
    tx_nonce_data: mpsc::Sender<NonceData>,
    context_mu: Arc<GpuContext>,
) -> impl FnOnce() {
    move || {
        for read_reply in rx_read_replies {
            let buffer = read_reply.buffer;
            // handle empty buffers (read errors) && benchmark
            if read_reply.info.len == 0 || benchmark {
                // forward 'drive finished signal'
                if read_reply.info.finished {
                    let deadline = u64::MAX;
                    tx_nonce_data
                        .clone()
                        .send(NonceData {
                            height: read_reply.info.height,
                            block: read_reply.info.block,
                            base_target: read_reply.info.base_target,
                            deadline,
                            nonce: 0,
                            reader_task_processed: read_reply.info.finished,
                            account_id: hex::encode(&read_reply.info.account_id),
                        })
                        .wait()
                        .expect("GPU worker failed to send nonce data");
                }
                tx_empty_buffers
                    .send(buffer)
                    .expect("GPU worker failed to cue empty buffer");
                continue;
            }

            // consume and ignore all signals
            if read_reply.info.len == 1 && read_reply.info.gpu_signal > 0 {
                continue;
            }

            gpu_transfer(
                &context_mu,
                buffer.get_gpu_buffers().unwrap(),
                *read_reply.info.gensig,
            );
            let result = gpu_hash(
                &context_mu,
                read_reply.info.len / 64,
                buffer.get_gpu_data().as_ref().unwrap(),
            );
            let deadline = result.0;
            let offset = result.1;

            tx_nonce_data
                .clone()
                .send(NonceData {
                    height: read_reply.info.height,
                    block: read_reply.info.block,
                    base_target: read_reply.info.base_target,
                    deadline,
                    nonce: offset + read_reply.info.start_nonce,
                    reader_task_processed: read_reply.info.finished,
                    account_id: hex::encode(&read_reply.info.account_id),
                })
                .wait()
                .expect("GPU worker failed to cue empty buffer");

            tx_empty_buffers.send(buffer).unwrap();
        }
    }
}
```

三． 区块头定义

The TAO 的区块头和区块定义位于 block.h，具体结构如下

```
class CBlockHeader
{
public:
    // header
    static const size_t HEADER_SIZE=4+32+32+32+4+4+32;
    static const int32_t CURRENT_VERSION=4;
    int32_t nVersion;
    uint256 hashPrevBlock;
    uint256 hashMerkleRoot;
    uint256 hashFinalSaplingRoot;
    uint32_t nTime;
    uint32_t nBits;
    uint64_t nNonce;
    std::vector<unsigned char> nSolution;

    // poc
    uint256 genSign;
    uint160 nPlotID;
    uint64_t nBaseTarget;
    uint64_t nDeadline;

    CBlockHeader()
    {
        SetNull();
    }

    ADD_SERIALIZE_METHODS;

    template <typename Stream, typename Operation>
    inline void SerializationOp(Stream& s, Operation ser_action) {
        READWRITE(this->nVersion);
        READWRITE(hashPrevBlock);
        READWRITE(hashMerkleRoot);
        READWRITE(hashFinalSaplingRoot);
        READWRITE(nTime);
        READWRITE(nBits);
        READWRITE(nNonce);
        READWRITE(nSolution);

        READWRITE(genSign);
        READWRITE(nPlotID);
        READWRITE(nBaseTarget);
        READWRITE(nDeadline);
    }

    void SetNull()
    {
        nVersion = CBlockHeader::CURRENT_VERSION;
        hashPrevBlock.SetNull();
        hashMerkleRoot.SetNull();
        hashFinalSaplingRoot.SetNull();
        nTime = 0;
        nBits = 0;
        nNonce = 0; //uint256();
        nSolution.clear();

        genSign.SetNull();
        nPlotID.SetNull();
        nBaseTarget = 0;
        nDeadline = 0;
    }

    bool IsNull() const
    {
        return (nBits == 0);
    }

    uint256 GetHash() const;

    int64_t GetBlockTime() const
    {
        return (int64_t)nTime;
    }
};

```

区块结构定义如下：

```
95
96  class CBlock : public CBlockHeader
97  {
98  public:
99      // network and disk
100     std::vector<CTransaction> vtx;
101
102     // memory only
103     mutable std::vector<uint256> vMerkleTree;
104
105     CBlock()
106     {
107         SetNull();
108     }
109
110     CBlock(const CBlockHeader &header)
111     {
112         SetNull();
113         *((CBlockHeader*)this) = header;
114     }
115
116     ADD_SERIALIZE_METHODS;
117
118     template <typename Stream, typename Operation>
119     inline void SerializationOp(Stream& s, Operation ser_action) {
120         READWRITE(*(CBlockHeader*)this);
121         READWRITE(vtx);
122     }
123
124     void SetNull()
125     {
126         CBlockHeader::SetNull();
127         vtx.clear();
128         vMerkleTree.clear();
129     }
130
131     CBlockHeader GetBlockHeader() const
132     {
133         CBlockHeader block;
134         block.nVersion       = nVersion;
135         block.hashPrevBlock  = hashPrevBlock;
136         block.hashMerkleRoot = hashMerkleRoot;
137         block.hashFinalSaplingRoot   = hashFinalSaplingRoot;
138         block.nTime          = nTime;
139         block.nBits          = nBits;
140         block.nNonce         = nNonce;
141         block.nSolution      = nSolution;
142
143         block.genSign = genSign;
144         block.nPlotID = nPlotID;
145         block.nBaseTarget = nBaseTarget;
146         block.nDeadline = nDeadline;
147
148         return block;
149     }
150
151     // Build the in-memory merkle tree for this block and return the merkle root.
152     // If non-NULL, *mutated is set to whether mutation was detected in the merkle
153     // tree (a duplication of transactions in the block leading to an identical
154     // merkle root).
155     uint256 BuildMerkleTree(bool* mutated = NULL) const;
156
157     std::vector<uint256> GetMerkleBranch(int nIndex) const;
158     static uint256 CheckMerkleBranch(uint256 hash, const std::vector<uint256>& vMerkleBranch, int nIndex);
159     std::string ToString() const;
160 };
161
```

四．零知识证明（zk-SNARKs）

零知识证明是指一方（证明者）向另一方（验证者）证明一个陈述是正确的，而无需透露该陈述是正确外的其他任何信息。

零知识证明分交互式和非交互式两大类：

- 交互式：证明者与验证者之间需要进行多轮通信；
- 非交互式：证明者按照协议向验证者发送一次消息，验证者根据协议即可验证，无需反复交互。

zk-SNARKs 是非交互式零知识证明的一种，全称 zero-knowledge succinct non-interactive argument of knowledge，即“简洁非交互零知识证明”。

- Succinct (简洁性) : 与实际计算的长度相比，生成的零知识证据消息很小，这对以字节为计量单位的区块链来说，极其重要；
- Non-interactive (非交互性) : 几乎没有任何交互。对于 zk-SNARK 算法来说，通常有一个构建阶段（Trusted Setup），构建阶段完成之后，证明者 (Prover) 只需向验证者 (Verifier) 发送一个消息即可。而且，SNARK 通常还有一个被称作是 "公开验证者" 的特性，意味着任何人无需任何交互即可验证零知识证据，这对区块链是至关重要的；
- Arguments (争议性) : 验证者只能抵抗计算能力有限的证明者的攻击。具有足够计算能力的证明者可以创建伪造的零知识证据以欺骗验证者 (这在现阶段是不可能的，其难度相当于破解椭圆曲线的私钥)。这也通常被称为 "计算完好性 (Computational Soundness)"，而不是 "完美完好性 (Perfect Soundness) "；
- of Knowledge : 对于一个证明者来说，在不知晓特定证明 (Witness) 的前提下，构建一个有效的零知识证据是不可能的。

zero-knowledge 前缀说明了在验证过程中，验证者除了知道证明者的陈述是正确有效的，不能学习到任何关于该论述的内容，比如，在 The TAO 中，矿工知道一笔交易是有效的，但是却不知道这笔交易的发起者，接收者以及转账金额等隐私信息。

其数学模型是：

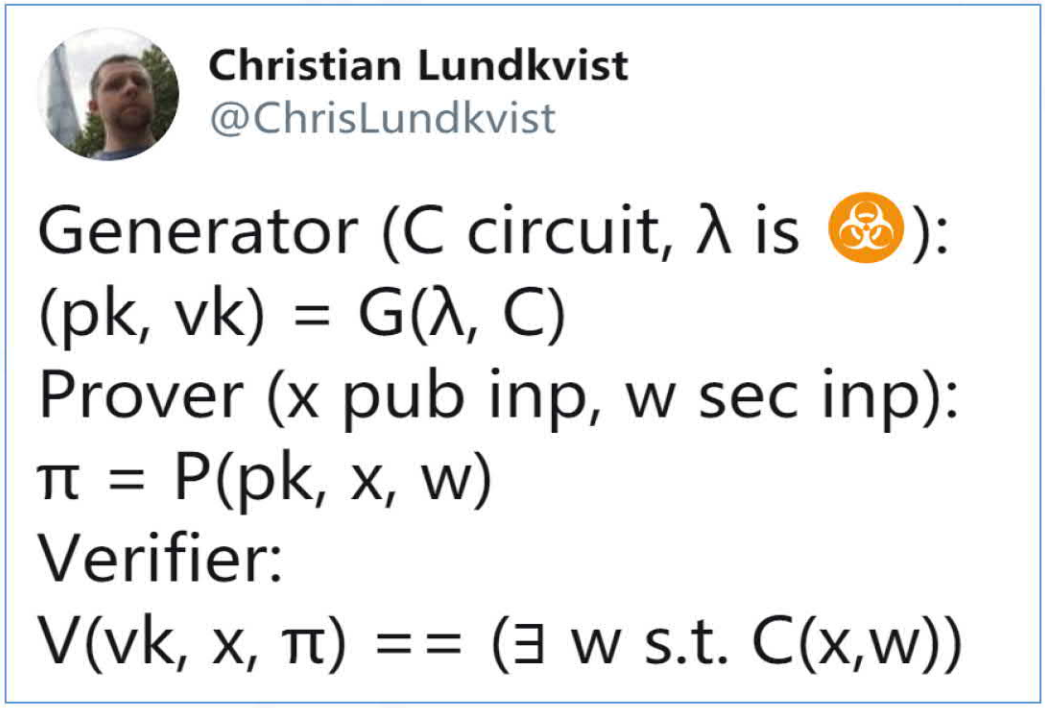
$$\Pr\left[\sigma \leftarrow \mathrm{Setup}(1^k) : \mathcal{A}^{\mathrm{Prove}(\sigma,\ldots)}(\sigma) = 1\right] \equiv \Pr\left[(\sigma,\tau) \leftarrow \mathrm{Sim}_1 : \mathcal{A}^{\mathrm{Sim}(\sigma,\tau,\ldots)}(\sigma) = 1\right]$$

其中，在 $(y,\omega) \in R\sigma$ 时，$\mathrm{Sim}(\sigma,\tau,y,\omega)$ 输出 $\mathrm{Sim}_2(\sigma,\tau,y)$或者均输出失败。

The TAO 通过 MPC（Multi-party Computation）协议生成 5 个参数，前 2 个参数分别是证明密钥（简称 pk，即 proving key）、验证密钥（简称 vk，即 verifying key），后面 3 个参数是电路参数。

pk 和 vk 是生成器 G 根据λ、C 来生成的。

λ：随机因子，也叫毒丸。



（来源：https://twitter.com/ChrisLundkvist/status/799807876982251520）

pk/vk 生成过程中，有多个参与者贡献自己的随机数，所有人共享的随机性的总和为λ。MPC 协议保证只要有一个参与者事后真正销毁了自己贡献的随机数，就能保证 pk 和 vk 的安全性，即无人能伪造证明。

C：电路变换函数，Circuit。

代码中用 r1cs 或者 cs 表示，rank-1 constraint system，一阶约束系统。

P：证明函数。

V：验证函数。

零知识证明的三步如下：

1、 G 根据输入λ、C 计算出 pk、vk。对于特定的区块链，pk、vk 只需要计算一次，并公开。

2、 证明人计算出证明值π = P(pk, 公开输入 x, 私密输入 w)。

3、 验证人通过计算 V(vk, 公开输入 x, π)为真，确认证明人持有私密输入 w 能使得 C(x, w)为真。

r1cs 参数文件用带命令行参数运行 taod 程序来生成：

taod -savesproutr1cs -experimentalfeatures

我们将举行盛大的 Trusted Setup 活动，为 zk-SNARKs 生成隐私证明参数。

五． 钱包

我们不重复发明轮子，我们的钱包兼容标准的 bip39/32/44 协议，即我们具备通用的助记词及分层确定性（HD）推导子地址协议，这在保证安全性的前提下，大大增加了便利性和易用性，并为手机端钱包兼容性打下良好基础。我们还将推出自己的硬件钱包，作为 PoC 生态的第一款硬件钱包，为社区的资产保驾护航。

我们的钱包地址分 2 类，以 7-开头的透明地址和以 tao-开头的隐私地址。所有地址均支持 HD 子地址推导。

六． 其他

The TAO 将为社区提供完备的工具链和使用链生态，从 p 盘程序到 miner 程序，从桌面版钱包到移动端钱包，从区块浏览器到矿池。且所有程序均将开源。

The TAO 首开先河，创造性地采用了源代码延时开源方式。我们将源代码打包并计算 SHA512，此哈希值被写入创世块中，此举既可以在 TAO 初期较脆弱的生态做到所有源代码延时半年左右开源以对抗直接抄袭，又可证明 Mainnet 从创世块产生之日起的代码纯净性。

经济模型

The TAO 采用通缩的经济模型，代币为 Utility Token，代号 TAO，无预挖。

总量：21,000,000 TAO

初始每块奖励：50 TAO

出块间隔：5 分钟

减半周期：210,000 块

攻防对策

任何的公链都面临各种不同类型的攻击，The TAO 也不会例外，为此我们做了如下攻防对策：

1. 矿霸攻击

PoC 生态，矿霸屡见不鲜。他们如蝗虫般到处野蛮掠夺 PoC 矿币，攫取短期经济利益，给生态带来严重破坏。对此，我们从技术上做了对抗策略。首先，是 p 盘参数，我们修改了 p 盘算法，从 8 字节 account ID 变为 20 字节，且并非简单数字扩充，这使得矿霸无法用已 p 的盘在 The TAO 早期全网算力不够大时，切大算力过来赚取利益；其次，主网将分为 pre-Mainnet 和 Mainnet 两个阶段，在 pre-Mainnet 阶段采取算力和区块高度综合考虑的方式保护生态初期和抵御矿霸攻击，详情见冷启动章节。

2. 51%攻击

PoC 有 PoW 类似的 51%攻击问题，为此，我们采用了最先进的算法来对抗。

3. 隐私泄露

我们时刻跟踪最新的 zk-SNARKs 以及 zk-STARKs 算法前沿，实时导入最新的研究成果，令 The TAO 保持健壮的稳定性和隐私性。

4. 冷启动

TAO 的主网分 pre-Mainnet 和 Mainnet 两个阶段，区别是全网算力(按 P 算), 100P 以内为 pre-Mainnet 阶段，超过 100P 及达到区块高度阈值，自动进入 Mainnet。

pre-Mainnet 以全网算力及区块高度决定矿工与基金会分配比例:

(0, 1P)　　&&　height <2016　　矿工:基金会　1:9

[1P, 10P)　&&　height<6048　矿工:基金会 2:8

[10P, 50P)　&&　height<14112　矿工:基金会 3:7

[50P, 100P)　&&　height<22176　矿工:基金会 5:5

[100P, ∞)　矿工: 100%

即全网算力达到 100P 时，正式进入主网阶段。

团队

The TAO 是匿名币，是 PoC 生态第一个匿名币，而我们也是来自世界各地的极客、白帽黑客组成的匿名团队。我们使用 ASM/C/C++的平均年限超过 15 年，我们对加密哈希算法、对称算法、不对称算法的研究和使用超过 20 年。我们分解过 RSA-512，也优化过 AES，我们用笔记本电脑构建过 MD5 碰撞（向王小云院士致敬），更不用说对于栈溢出、堆喷射这类经典但在今天并不过时的漏洞挖掘技能的掌握。我们有 nsfocus，kanxue 的“创世”成员，也不乏 irc 世界里 hacker scene、cracker scene 的优秀 ID。

虽系出名门，但我们并不想让背后的光圈成为 The TAO 的包袱，我们更乐意用技术，用密码学来回馈社区，回馈社会。我们的邮箱地址是 thetao[at]riseup[dot]net。

Don’t call us, we will call you if you are special enough.

社区

生态的健康、稳定、可持续发展离不开社区，我们看重社区，尊重社区，core team 将与社区共同引导和促进 The TAO 的发展，PoC 的布道者们已经为我们打下一片小天地，在感激他们的同时，我们向全世界公开招募社区组织者和参与者。

让我们携手共建，共创 PoC+zk-SNARKs 的辉煌！

路线图

The TAO 将直接以 Mainnet 方式面世，做 PoC 的隐私币，乃至整个世界的隐私币。

下一步，我们会以每 6 个月一个大版本的迭代速度完成与隐私即时聊天工具的融合，与 NFT 的融合，与 DeFi 的融合，与分布式预言机的融合。大力推广隐私币，实现真正意义上的移动端零知识证明的隐私币钱包，基于 PoC 的隐私稳定币等等。

在 2021 年 Q1，我们将登陆 DEX，Q4 登陆 1~2 个知名大交易所。

一切美好，都在悄然发生，敬请期待……

致谢

感谢中本聪，王小云，小 v，Zcash Core Team，Burst Core Team，BHD Core Team &Community，以及无数默默支持我们的亲朋好友！

Love you, forever!

引用

https://bitcoin.org/en/bitcoin-paper

https://ethereum.org/learn/

https://z.cash/technology/

https://www.burst-coin.org/

http://www.btchd.org/