A Proof-of-Stake scheme for confidential transactions with hidden amounts

sowle

Zano project
val@zano.org
https://zano.org

July 2021*

Abstract

This article explores a way of implementing a Proof-of-Stake mining algorithm in an environment where amounts are hidden with homomorphic commitments.

We propose an algorithm which is compatible with such transactions, including transactions with mixed-in decoys.

1 Notation

Let $\mathbb{G}$ denote the main subgroup of the Ed25519 curve ([1]) and $\mathbb{Z}_p$ denote a ring of integers modulo p.

l is the order of $\mathbb{G}$: $l = \#\mathbb{G} = 2^{252} + 27742317777372353535851937790883648493$.

For any set X, $x \xleftarrow{\$} X$ means uniform sampling of x at random from X.

For any integers x, y, $\left\lfloor \frac{x}{y} \right\rfloor$ denotes the integer part of *integer arithmetic* division.

2 PoS scheme using open amounts

In this section we describe how PoS mining was originally implemented in Zano.

Suppose Alice has some unspent outputs and wants to mine a PoS block using one of them as a stake. In such a scenario she acts as follows (Fig. 2.1):

1. Gets the hash identifier of the last PoW block in the blockchain, *last_pow_id*.
2. Gets the last PoS block in the blockchain and gets the stake kernel hash identifier from it, *last_pos_kernel_id*. Together with *last_pow_id* they are called the "stake modifier". It changes each time a new block is added to the blockchain.

*Version 3.1. Last update: 2021-08-17. Check here for the latest version.

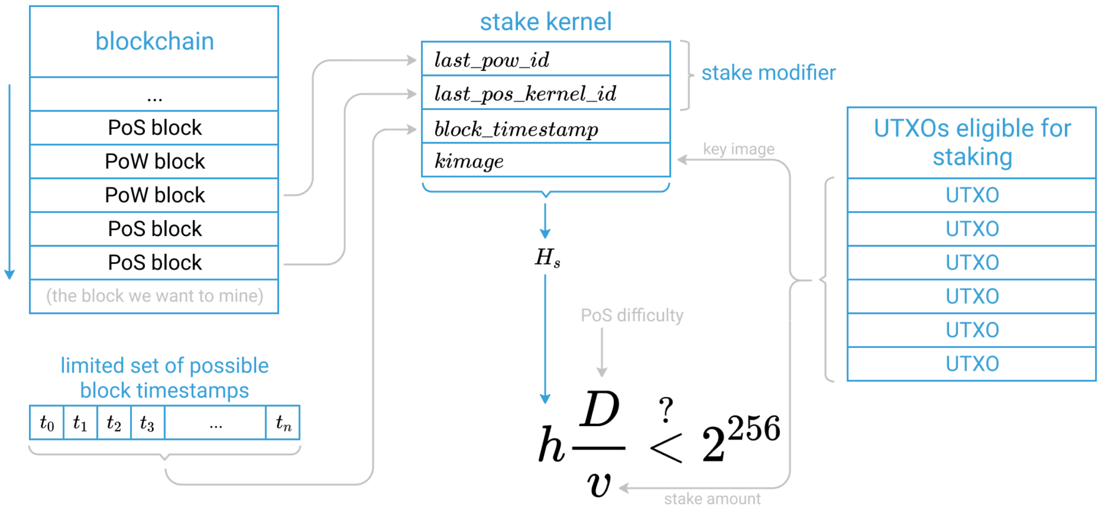

Fig. 2.1. Scheme of the original PoS mining process, as originally implemented in Zano

3. Makes set T of all possible timestamps for the new PoS block:

$$T = \{t : t_{min} \leq t \leq t_{max},\ t \equiv 0 \mod 15\}$$

where t_{min} and t_{max} are bound to the current blockchain conditions. For the sake of simplicity we can assume that

$$t_{min} = \tau - T,\ t_{max} = \tau + T$$

where T is a constant and τ is the current timestamp established in such a way, that it is the same across all the network nodes.

4. Makes set U of all her unspent transaction outputs (UTXO) that are eligible for staking (i.e. not locked, mature enough etc.). For each output u from U she also precalculates the key image I_u.

5. Each pair $(t, u) \in T \times U$ is checked against the PoS winning condition as follows:

   (a) for output u build *stake kernel* K_u as a concatenation:

   $$K_u = last\_pow\_id \parallel last\_pos\_kernel\_id \parallel t \parallel I_u$$

   where I_u is the key image of the stake output u;

   (b) Calculate hash $h_u = H_s(K_u)$;

   (c) Finally, check the main condition:

   $$h_u \frac{D}{v_u} \overset{?}{\leq} 2^{256} \tag{1}$$

   where D is the current PoS difficulty, and v_u is the amount of the stake output u.

If inequality (1) holds then it means the success of PoS mining! A block with timestamp t and stake input, spending output u, can be constructed and broadcast to the network.

If for all pairs (t, u) the (1) does not hold, Alice needs to wait until one of the following happens:

- a new block is added to the blockchain (this will change either *last_pow_id* or *last_pos_kernel_id*);
- some time passes (this will change t_{min} and t_{max}).

Once this happens, Alice can perform another attempt at mining (items 1-5) as all K_u, and thus h_u, will have different values, giving new opportunities to meet the main condition.

We'd like to note the following important property of (1): as h_u is the result of cryptographic hash function H_s and could be considered as distributed evenly over $\mathbb{Z}_l$, the probability of meeting the main condition is proportional to output amount v_u.

3 PoS direct spending scheme using hidden amounts

(*Note: this variant of the scheme requires stake inputs to directly refer to their outputs, i.e. without decoys.*)

Consider a hidden amount scheme, where amount v of an output is hidden using Pedersen[1] commitment A:

$$A = vG + fH \quad (v < 2^{64},\ f \neq 0)$$

where G and H are generators in $\mathbb{G}$ for which DL relation is unknown, and f is a random hiding mask.

It's easy to see that (1) can't be used anymore because it requires using non-hidden v. Let's see how the main inequality could be modified.

Suppose Alice has already prepared sets of timestamps (T) and outputs (U) eligible for staking as mentioned in section 2. She then considers each pair $(t, u) \in T \times U$ against the PoS winning condition. She calculates

$$h = H_s(\mathit{last\_pow\_id} \parallel \mathit{last\_pos\_kernel\_id} \parallel t \parallel I_u)$$

As we mentioned above, h can be considered as uniform randomness distributed evenly over $\mathbb{Z}_l$. Because hiding mask $f \neq 0$ and it is fixed for the selected output u, the multiplication $hf \pmod{l}$ can also be considered as uniform randomness over $\mathbb{Z}_l$. TODO: formal proof may be needed.

Taking this into account, Alice checks the slightly adjusted main inequality:

$$hf \bmod l \ < \ \left\lfloor \frac{l}{D} \right\rfloor v \tag{2}$$

where l is the order of the main subgroup. Here we moved from 2^{256} (used originally in (1)) to l as all scalar operations in all the following equations hold modulo l except the division $\left\lfloor \frac{d_0}{D} \right\rfloor$.

[1]More information in original paper by T.P. Pedersen: 3.

Note that as soon as $D > 2^{64}$ and $v < 2^{64}$, the right side of (2) never overlaps the l.

Now we transform the inequality to equality:

$$hf \mod l = \left\lfloor \frac{d_0}{D} \right\rfloor v - b_v, \quad D \le d_0 \le l,\ b_v < 2^{64} \tag{3}$$

Once (2) holds, Alice needs to calculate d_0 and b_v so that (3) holds. If then she can convince a verifier of knowing d_0, b_v to be in corresponding ranges, she can also convince them that (2) holds[2] for particular h, and thus, for pair (u, t). Now we construct such a proof in a NIZK-manner.

Rewriting (3) slightly:

$$(3) \quad \Leftrightarrow \quad hf - dv + b_v = 0, \quad d = \left\lfloor \frac{d_0}{D} \right\rfloor \tag{4}$$

Let $b_f = df - hv$. The following equality holds:

$$hv - df + b_f = 0 \tag{5}$$

Use (4) and (5) as scalar parts for scalar multiplication with G and H correspondingly:

$$(4), (5) \Rightarrow \begin{cases} hf - dv + b_v = 0 & | \times G \\ hv - df + b_f = 0 & | \times H \end{cases} \tag{6}$$

Considering commitments $A' = fG + vH,\ A = vG + fH,\ B = b_vG + b_fH$, we can rewrite (6) in terms of group element operations:

$$hA' - dA + B = \mathbf{0} \tag{7}$$

where $\mathbf{0}$ is the identity element of $\mathbb{G}$.

3.1 A' proof

As part of the whole PoS proof, Alice needs to convince a verifier that $A' = fG + vH$ without revealing v and f. As the verifier knows public $A = vG + fH$ we can construct a Schnorr-like proof as follows:

1. Alice generates randomnesses $r_0, r_1 \xleftarrow{\$} \mathbb{Z}_l$
2. Calculates $R_0 = r_0(G + H), R_1 = r_1(G - H)$
3. Calculates non-interactive challenge $c = H_s(R_0, R_1, A', A)$
4. Calculates $y_0 = r_0 + c(v + f)$, $y_1 = r_1 + c(v - f)$
5. Sends (c, y_0, y_1) to the verifier.

[2]Except with negligible probability in case when $\left\lfloor \frac{d_0}{D} \right\rfloor v < b_v$

6. Verifier makes sure that
$$c \stackrel{?}{=} H_s(y_0(G+H) - c(A+A'), y_1(G-H) - c(A-A'), A', A)$$
If the above equation holds, the verifier is convinced that $A + A' = k_0(G+H)$ and $A - A' = k_1(G-H)$ for some k_0 and k_1, and thus due to Lemma 1 he is convinced that $A' = fG + vH$.

In appendix B we give an intuition for the fact that this proof does not reveal parts of commitments.

3.2 PoS proof

Let's summarize the whole scheme.

1. Alice prepares a set of possible timestamps T and staking outputs U.
2. For each pair (t, u) she calculates $h = H_s(\dots)$ and checks it against the winning condition (2).
3. If (2) holds, she calculates:
$$\begin{aligned} A' &= fG + vH \\ d &= \left\lfloor \frac{hf \bmod l}{v} \right\rfloor + 1 \\ b_v &= dv - hf \\ b_f &= df - hv \\ B &= b_v G + b_f H \end{aligned}$$
4. Generates proof (c, y_0, y_1) for the fact that $A' = fG + vH$ (section 3.1).
5. Makes a PoS block with stake output u and timestamp t, and adds PoS proof σ to the block's data:
$$\sigma = \{d, A', (c, y_0, y_1), B, \mathcal{R}_B\} \tag{8}$$
where $\mathcal{R}_B$ is a range proof (e.g. Bulletproofs) for the fact that $b_v < 2^{64}$

Verifiers on the network check the PoS block as follows:

1. $0 < d \leq \lfloor \frac{l}{D} \rfloor$
2. Check Schnorr-like signature (c, y_0, y_1) for A'.
3. Check $hA' - dA + B = \mathbf{0}$
4. Check range proof $\mathcal{R}_B$

3.3 Limitations

The proposed scheme works only under the following conditions:

- Proof-of-stake difficulty: $D > 2^{64}$
- Output's amount: $v < 2^{64}$
- Commitment's mask: $f \neq 0$

3.4 Size of PoS proof

Let's estimate the size of the proof (8).

It has two group elements, four field elements plus the size of the range proof $\mathcal{R}_B$.

In the case of using Bulletproofs+ [2] the size of a single proof with $n = 64$ is

$$2 \cdot \lceil \log_2(n) \rceil + 3 = 15$$

group elements and 3 field elements.

In total, for the PoS proof we have 17 group elements and 7 field elements. If both field and group elements have a compressed size of 32 bytes, which is the case for Ed25519 used in Zano, then the total size of the proof can be estimated as $17 + 7 = 24$ elements or $24 \cdot 32 = 768$ bytes.

4 Ring-friendly PoS hidden amount scheme

The solution we proposed in section 3 can't be used in the case of a CT-like mining transaction with a non-empty decoy set: in such a transaction the stake input would refer to a *set* of outputs and thus a set of *pseudo* output commitments is used instead of a single commitment A in which the input's hidden amount v is committed to. Therefore, verifiers on the network would not be able to check (7) as they don't know the particular A.

Here we propose a solution to this problem.

4.1 Ring-friendly PoS scheme

Suppose Alice wants to mine a PoS block, and she already went through steps 1-3 above (subsection 3.2). She calculated such A', d, B that $hA' - dA + B = \mathbf{0}$ holds.

Then she continues as follows:

4. Randomly selects a set of apparently unspent decoy outputs $\{u_i\}$ from the blockchain and puts her output, which met the PoS main condition (2), at random index π of that set. Note that the i-th decoy output has its hidden amount committed to in A_i (and A_π is the commitment to her own output). Also note that in general Alice doesn't know amounts v_i and masks f_i for the outputs she selected as decoys.

5. For each decoy output u_i $(i \neq \pi)$ Alice generates random values $b_{v,i} \xleftarrow{\$} \mathbb{Z}_{2^{64}}$, $b_{f,i} \xleftarrow{\$} \mathbb{Z}_l$ and commitments to them:

$$B_i = b_{v,i}G + b_{f,i}H \tag{9}$$

She also calculates complementary commitment A':

$$A'_i = h^{-1}(dA_i - B_i) \tag{10}$$

As before, B_π and A'_π were calculated earlier and correspond to her own output.

Now, consider the system of equations $hA'_i - dA_i + B_i = \mathbf{0}$ or in other notation:

$$\begin{cases} hA'_0 & -dA_0 & +B_0 & = \mathbf{0} \\ & \dots & & \\ hA'_\pi & -dA_\pi & +B_\pi & = \mathbf{0} \\ & \dots & & \\ hA'_{n-1} & -dA_{n-1} & +B_{n-1} & = \mathbf{0} \end{cases} \tag{11}$$

Here the π-th equation holds, because of equivalence to (7), and others hold because of (9) and (10).

6. Consider a pair of group elements $(A_i + A'_i, A_i - A'_i)$. If $i = \pi$ Alice is able to calculate secrets k_0 and k_1 so the following holds:

$$\begin{cases} A_i + A'_i = k_0(G + H) \\ A_i - A'_i = k_1(G - H) \end{cases}$$

Indeed: $k_0 = v+f$, $k_1 = v-f$. For all the others $i \neq \pi$ Alice would not be able to calculate such k_0, k_1, unless she selected her own output as a decoy (and thus, she knows the corresponding hidden amount and mask). But in such a case (11) holds only if she is able to calculate appropriate d and $b_{v,i}$ as well, which is equivalent to satisfying (2) for that "decoy" output.

According to Lemma 1, a proof of knowing k_0, k_1 is equivalent to proof that $A'_i = fG + vH$.

Alice adds proof of knowing secret keys k_0, k_1 into the main ring signature as two additional layers[3], in order to convince verifiers that $A'_i = fG + vH$ and $A_i = vG + fH$ both hold for the output being spent.

To achieve this, we can extend the main ring signature by adding two more group elements to the calculation of the non-interactive challenge as follows:

$$\begin{aligned} c_{\pi+1} &= H_s(\dots,\ \alpha_0(G+H),\ \alpha_1(G-H) \\ c_{i+1} &= H_s(\dots,\ r_i^0(G+H) + c_i(A_i + A'_i),\ r_i^1(G-H) + c_i(A_i - A'_i) \\ r_\pi^0 &= \alpha_0 - c_\pi k_0 \\ r_\pi^1 &= \alpha_1 - c_\pi k_1 \end{aligned}$$

7. Finally, Alice adds PoS signature $\sigma = \{d, \{A'_i\}, \{B_i\}, \{\mathcal{R}_{Bi}\}\}$ to the mining transaction, where $\{\mathcal{R}_{Bi}\}$ are range proofs (e.g. Bulletproofs+) for the fact that $b_{v,i} < 2^{64}$.

4.2 Verification of ring-friendly PoS scheme

Verifiers on the network check the PoS block as follows:

1. $0 < d \leq \lfloor \frac{l}{D} \rfloor$
2. Calculate h and check $hA'_i - dA_i + B_i = \mathbf{0}$

[3]Here we're using terminology and ideas from Multi-layered Linkable Spontaneous Anonymous Group signature proposed in 4.

3. Check stake input's ring signature with additional layers for A'_i
4. Check range proofs $\mathcal{R}_{Bi}$

4.3 Size of ring-friendly PoS proof

Let's estimate the size of the proof for $n-1$ decoy outputs, where the total size of the ring is n. Assume we're using aggregated Bulletproofs+ for range proofing. According to [2] it comprises $2 \cdot \lceil \log_2(m) + \log_2(n) \rceil + 3$ elements in $\mathbb{G}$ and 3 elements in $\mathbb{Z}_l$, where $m = 64$ for range 2^{64}.

For each ring member we need to store 2 elements in $\mathbb{G}$ (A'_i and B_i) and, supposedly, only 2 elements in $\mathbb{Z}_l$ for ring signature extension (r_i^0 and r_i^1).

Additionally, we need to store one element in $\mathbb{Z}_l$ per PoS signature (d).

In total we have $2n + 2 \cdot \lceil \log_2(n) \rceil + 15$ group elements and $2n + 4$ field elements. If both field and group elements have a compressed size of 32 bytes, which is the case for Ed25519 used in Zano, then the total size of additional PoS data can be estimated as $4n + 2 \cdot \lceil \log_2(n) \rceil + 19$ elements or $128n + 64 \cdot \lceil \log_2(n) \rceil + 608$ bytes.

References

[1] Daniel J. Bernstein et al. *Ed25519: high-speed high-security signatures.* https://ed25519.cr.yp.to.

[2] Heewon Chung et al. *Bulletproofs+: Shorter Proofs for Privacy-Enhanced Distributed Ledger.* https://eprint.iacr.org/2020/735.

[3] T.P. Pedersen. *Non-interactive andinformation-theoreticsecureveriØablesecretsharing.Advancesin Cryptology-CRYPTO'91.* 1991.

[4] Adam Mackenzie Shen Noether and Monero Core Team. *Ring Confidential Transactions, MRL-0005.* https://web.getmonero.org/resources/research-lab/pubs/MRL-0005.pdf. 2016.

A Lemmas

Lemma 1. *Let $v, f \in \mathbb{Z}_l$ and $A = vG + fH$, where G and H – public generators in cycle group $\mathbb{G}$ of prime order l, for both of which DL relation is unknown. The Prover knows v and f. The Verifier knows only A. If the Prover wants to convince the Verifier that the given $A' = fG + vH$ without revealing secrets v and f, it is enough to provide a proof to the fact that he knows y_0 and y_1 such that:*

$$A + A' = y_0(G + H)$$
$$A - A' = y_1(G - H)$$

Proof. Suppose the Verifier is convinced that the Prover knows y_0 and y_1 such that the equations above hold. Also suppose that, contrary to the lemma's statement, $A' = aG + bH \neq fG + vH$ where $a, b \in \mathbb{Z}_l$.

Substitute equations for A and A':

$$\begin{cases} vG + fH + aG + bH = y_0(G + H) \\ vG + fH - aG - bH = y_1(G - H) \end{cases}$$

As DL relation between G and H is unknown, we can split the equations:

$$\begin{cases} (v + a)G = y_0G \\ (f + b)H = y_0H \\ (v - a)G = y_1G \\ (f - b)H = -y_1H \end{cases}$$

$$\Rightarrow \begin{cases} v + a = f + b \\ v - a = b - f \end{cases}$$

$$\Leftrightarrow \begin{cases} 2v = 2b \\ 2a = 2f \end{cases} \Rightarrow A' = aG + bH = fG + vH$$

This contradiction concludes the proof.

$\square$

B Intuition for A' proof

Let us give an intuition for the fact that the A' proof given in subsection 3.1 for the fact that $A' = fG + vH$ does not reveal vG, fH, fG or vH to the Verifier. TODO: formal proof may be needed

The Verifier gets $\{y_0, y_1, c\}$ from the Prover and thus can construct the following system:

$$\begin{cases} A = vG + fH \\ A' = fG + vH \\ y_0 = r_0 + c(v + f) \\ y_1 = r_1 + c(v - f) \\ c = H_s(r_0(G + H), r_1(G - H), A, A') \end{cases}$$

where all the known values are on the left (except the generators). As c is the result of the cryptographic hash function H_s of fixed arguments it can be treated by the Verifier as a known constant, so we'll exclude the equation for c.

Let's multiply all terms in equations for y_0 and y_1 by G and H, and make a substitution:

$$\begin{cases} y_0G = r_0G + \alpha + \gamma \\ y_0H = r_0H + \delta + \beta \\ y_1G = r_1G + \alpha - \gamma \\ y_1H = r_1H + \delta - \beta \\ cA = \alpha + \beta \\ cA' = \delta + \gamma \end{cases} \tag{12}$$

where

$$\alpha = cvG \quad \beta = cfH \quad \gamma = cfG \quad \delta = cvH$$

We would like to show that it's impossible for the Verifier to calculate $\alpha, \beta, \gamma, \delta$.

There are six linearly independent equations in (12) and also six unknown values: $\alpha, \beta, \gamma, \delta, r_0, r_1$. It looks like (12) could be solved.

However, due to discrete logarithm assumption scalar multiplications r_0G, r_0H, r_1G and r_1H should be considered as four independent unknowns rather than two r_0, r_1. Therefore, there are eight unknowns in (12) and it can not be solved.

Note that if we assume that $r_0 = r_1$ or at least $r_0k = r_1$, where $k \in \mathbb{Z}_l$ is a known value, the number of unknowns would be reduced to six and the system could be solved, so the Verifier could calculate $\alpha, \beta, \gamma, \delta$.

However, the Prover generates r_0 and r_1 uniformly at random in $\mathbb{Z}_l$.

C How to ensure $f \neq 0$

Here we discuss how to ensure that $f \neq 0$ for commitments in case the underlaying transaction protocol does not guarantee that.

In subsection 3.3 we mentioned that the proposed PoS scheme works only under certain conditions, and one of them is the commitment mask f must be nonzero for all staking outputs. The reasoning behind that is simple: suppose Alice prepared a UTXO with amount v committed to A with zero mask f: $A = vG + 0H = vG$. Such an output can be staked instantly as the winning condition is met regardless of h:

$$hA' - dA + B = \mathbf{0} \quad (7) \quad \overset{f=0}{\Longrightarrow} \quad hvH - dvG + B = \mathbf{0}$$

This implies that for *any* given h, point $B = (dv)G + (-hv)H$ and scalar $d = 1$ will satisfy all conditions mentioned in subsection 4.2.

Obviously, this can be solved by providing a proof that $f \neq 0$ for each output in the blockchain and forcing all verifiers to validate it. To avoid such undesirable expenses consider the following idea.

Let's require adding $f'H$ to all the commitments A_i before using them in the PoS protocol, where f' is a public constant that is unknown to Alice when she generates hiding mask f for the output.

For instance, f' can be calculated as $f' = H_s(block\_id)$, where $block\_id$ is the hash identifier of the block containing that output. Consequently, substituting $f + f'$ for f in (6) we get:

$$\begin{cases} h(f + f') - dv + b_v = 0 & | \times G \\ hv - d(f + f') + b_f = 0 & | \times H \end{cases}$$

Now for (7) we obtain:

$$hA' - dA + B + f'(hG - dH) = \mathbf{0}$$

With such a modification Alice won't be able to gain an advantage when choosing f.