QRL Proof-of-Stake Algorithm

info@theqrl.org

July 6, 2017

1 Introduction

What follows is a brief description the proposed proof-of-stake algorithm chosen by the Quantum Resistant Ledger project. This algorithm and the document is a work in progress and may be subject to change.

The algorithm features:

1. Staking with no minimum security deposit.
2. Use of hash-chain authentication in lieu of signatures
3. Probabilistic and random novel block selection method utilising iterative keyed hash-chains.
4. Minimal hardware requirements

2 Overview algorithm

We are following some of the idea's described in http://vitalik.ca/files/randomness.html, under section Hybrid RANDAO / Private randomness. The basic idea is the following:

- All participants construct hash-chains, which are used to have random values.
- These random values are, together with the stake, used to determine the block winner
- To counter pre-mining attacks (optimizing the hash-chain by an attacker) a public seed is constructed before epoch transition, using randomness of all participating stake validators
- This seed is updated after each block, "resetting" the randomness for all participants after each block and preventing any validator guessing the outcome more than one block into the future.

- To protect against long range / hash-replay attacks (constructing a completely new chain), multiple hash-chains are used to commit stake validators to only one chain.

We will now go into more details.

2.1 Commitment phase

Each participants who wishes to participate in staking in the next epoch, lasting k blocks, can send a stake-transaction in the current epoch, where $k = 10000$ at this moment. The stake transaction contains the following:

1. The end of a long hash-chain A, which means a value $h_A = H^{k+1}(nonce_A)$, where $nonce_A$ is some random number chosen by the staker.
2. A set of other long, hash-chains B, which means values $h_B = H^k(nonce_{B_i})$, where $nonce_{B_i}$ are random numbers chosen by the staker. We currently take 50 hash-chains in set B although this number may rise pending a security assessment.

The stake participants save all $nonce_A, nonce_{B_i}$ for their secret value. This stake transaction can only be done before the last ℓ blocks in the current epoch, for example by taking $\ell = 1000$ we do not allow any more stake transactions to be added in the last 1000 blocks. New stake transactions with a commitment similar to one in another address are discarded.

2.2 Determining stake

There is a stake snapshot at the start of the epoch and that stake is what is used as weight to determine winners in the block selection process. Transactions during the epoch will not contribute to stake calculations and funds are frozen for active stake validators during the epoch.

2.3 Reveal phases

2.3.1 Initializing the public random seed

To prevent pre-mining attacks, a public seed is determined before the epoch transition. In the last ℓ blocks of the current epoch, all participants need to reveal the first value in hash-chain A, i.e. $H_A^k(nonce_A)$. The initial seed for the next epoch S_0 will be the XOR of all values, i.e.

$$S_0 = \bigoplus_{p \in P} h_p$$

for all first hash reveals h_p of partipant p in staker set P (the set of all stakers). Note that the order does not matter in which the reveals are XORed. The seed will be updated later in the epoch, and we name the seed for block j in the epoch S_j.

2.3.2 Preventing seed manipulation

Whilst the order in which reveals are received and XORed together to generate the public seed does not matter, an attacker may attempt to manipulate the seed by:

1. failing to reveal
2. waiting until honest validators have revealed and altering the seed with validator reveals in his control

To avoid this problem the following is proposed:

1. failure to reveal results in loss of all security deposit
2. valid reveal votes are only accepted up to a certain distance z, within l, based upon stake / total staked for each validator

   i.e. accounts with lowest stake must declare earlier

2.3.3 Hash-chain A

To determine the winner of block j in the epoch, each participant reveals $H_A^{k-j}(nonce_A)$, i.e. the previous value in hash-chain A. These values are checked with previous commitments or the end of the hash-chain saved in transaction. It is possible for stake validators to be offline for some period, but each participant has to be revealing the correct index of the hash-chain. Each participants waits for a certain a mount of time and collects all reveals from participants.

2.3.4 Hash-chains set B

To prevent hash-chain A reveal hash replay attack each participant also reveals a value t (so t is between 1 and 50) in one of the chains of set B, depending on the previous blockheader hash and their stake address ($hash(headerhash||stakeaddress)$. Thus the participant also reveals $H_B^{k-j}(nonce_{B_t})$ for block j and chain t.

2.4 The block winner

The winner of block j is the participant who revealed the hash of chain A, XORed with the current seed S_j, with the lowest value, accounted for by stake. This means participant P_1 with hash reveal $R(P_1) = h_A^{k-j}(key||nonce)$ and balance $bal(P_1)$ out of total balance $T = \sum_a bal(a)$, should have some weighted value. The formula we use to calculate this value, where $N = 256$ output bits of the hash-function, is the following, for block j in the epoch:

$$F(P_1, j) = -\log_2((R(P_1) \oplus S_j)/2^N) \cdot \frac{1}{bal(P_1)} = \frac{1}{bal(P_1)}(N - \log_2(R(P_1) \oplus S_j))$$

You can show that in this case, $Pr[F(P_1, j)\colon \texttt{ lowest value}] = \frac{bal(P_1)}{T}$, which is the behaviour we want. Details why are given in: http://bitfury.com/content/5-white-papers-research/pos-vs-pow-1.0.2.pdf, appendix B. We will add a mathematical proof in a later version of this document why this gives a fair distribution.
The seed S_j will also be updated by the hash-reveal of the winner of the block, by setting $S_j = hash(S_{j-1}||R(P_1))$. Because our block selection algorithm is both random and probabilistic it negates the potential Sybil attack from attacker stake validator accounts being effective.

The winner of the block signs the block-header which contains a merkle tree root hash of the transaction list, a NTP corrected timestamp and the list of all reveals by stake validator participants. The XMSS signature is added to the block to validate correctness. Newly minted coins are allocated to the winner.

It is possible that, due to network delay, participant P_1 thought he won and added a block, but actually participant P_2 won. Everyone should discard the block from P_1 and instead build upon P_2's block if the weighted hash value of P_2 is lower.

2.5 Block-reward

Currently all block reward is accrued by the winning stake validator with the lowest published hash-value. As there is no mathematical gain to be obtained by an attacker by using Sybil stake validator addresses there is no need to limit security deposit requirements for staking.

3 Security Analysis

A vital assumption in proof-of-stake models is assuming honest nodes hold at least 51% stake and are online and participating in staking. This is also required in the security analysis.

3.1 Follow the correct chain

3.1.1 Building on next block

Honest nodes should always build upon the latest valid block containing the lowest weighted value from hash-chain A. Usually this will result in a chain composed of blocks containing consecutive winning hash-chain A weighted reveal values.

In practice the node implementation currently uses a matrix of received blocks (a buffer) which sits in memory and is x blocks in length, sat atop the chain.

3.1.2 Buffer rules

1. Always build upon block sequence with consecutive lowest adjusted hash-chain A value whilst in the buffer (x blocks deep) - this may result in nodes discarding later blocks in the buffer and building upon a 'stronger' but shorter chain on occasion when a block arrives late.

2. New blocks arriving deeper than x are ignored.

3.1.3 Fork rules

True chain forks may only occur which are more than x blocks deep and less than y blocks from the current blockheight (assuming signatures are correct on both sides of the fork), where x is the length of the buffer and y is the number of blocks of replay attack protection offered by B set of hash-chains.

1. For a fork (deeper than x, above y blocks) choose the side of the fork with the most cumulative stake in the reveals in the blockheader of each block.

2. Nodes should not consider forks: deeper than y, which cross an epoch transition, or which grow the chain forward into the future fraudulently (assessed by NTP-corrected blockheader timestamps).

New stakers should always build on top of the chain with highest stake score as per 1.

3.2 Double spend

Conventional double spending attempts may occur by displacing one transaction with another different transaction which is confirmed earlier into a block.
To give certainty that the transaction history of the chain may not change by a fork up to depth y blocks, it is necessary then to wait for y confirmations.

3.3 Nothing at stake

When a fork happens, honest nodes should build on top of the strongest chain by sequence of stake-weighted hash-chain A reveal. However, there is no theoretical reason for nodes to not also try to build on top of weaker permutations. As new blocks require signatures from winners, it is possible for the whole network to see a node trying to build on top of a wrong chain. It is possible to put a penalty on this, for example all coins of that attacker can be transferred to all honest nodes, creating an incentive to behave honest but also another incentive to participate in staking.

The authors feel this attack is theoretical as honest node behaviour in the event of two reveal votes arriving from a particular stake validator is to discard the second vote automatically.

3.4 Long range attacks / Hash replay attack

In this case, an attacker tries to make a completely different fabricated chain in his favour from an earlier blockheight.

This will be non-trivial to achieve for the following reasons:

1. The attacker must create a chain to the current blockheight with valid winning blocks (each containing a valid XMSS signature of the blockheader from the victorious stake validator).

2. As the epoch progresses the hash indexes of earlier hash-chain A reveals are known by the network. To prevent earlier stake validator votes being replayed to falsely demonstrate stake validator support for the attacker fork, a second reveal hash for each validator (hash-chain B reveal hash) must also be supplied. Given we set a reorg limit of y blocks we can be sure that this strategy is unlikely to succeed. The only mitigation the attacker can achieve is to remove stake validator reveal votes he is unable to replay into the forged block - but this will result in less total amount of committed stake per block, which in turn means honest nodes will continue to follow the correct/original chain during a fork recovery process.

As the honest nodes will accept a fork less then x blocks deep, the attacker wants to construct a fork of at least x blocks deep. Furthermore, he needs to win each block honestly for two reasons: to get more cumulative stake weight and to be able to construct a valid signature, as he does not get that from other winners. Assuming the attacker has about 50% stake (this means we assume the strongest attacker possible), his probability of winning all x block equals $(1/2)^x$. Having the correct hash reveal of chain B, which we take 50 chains, of k stakers, equals at most $\left(\frac{x}{50}\right)^k$. So the total success probability of attacking the chain and "stealing" the hash-reveals of an honest staker equals

$$(1/2)^x \cdot \left(\frac{x}{50}\right)^k$$

By taking $x = 10$ blocks deep buffer and only $k = 10$ stakers, we get a success probability of 2^{-33}. In other words, the attacker has to compute 2^{33} hash-values (similar to mining) to get a correct malicious blockchain. This will take significant amount of time on current computers. On top of that, the honest chain will also move forwards, which means he needs to do this attack faster than one new block creation. We do not see this possible. In the future, both value x and number of chains in set B can be increased easily for improved security.

3.5 Block withholding attacks

A block withholding attack is an attack in which the block winner does not reveal that he won the block, but withholds it to his advantage instead. A block withholding attack in our protocol cannot occur by keeping the block more than x blocks, as after that no reorg changes are allowed to the main chain which then becomes immutable. Witholding a winning reveal hash costs the attacker the block reward of the block. Furthermore, our protocol instructs nodes which believe they are close to the winning vote to also produce blocks which may be selected if the true winner fails to construct a block in a timely manner.

3.6 Pre-mining (of hash-chain A)

You might think that someone can pre-mine his hash-chain to get highest probability of winning a block. However, there are two main reasons why this fails. Since the initial seed S_0 is constructed by using randomness of all participants, no attacker can account for what this seed will be. This means an attacker cannot know how to optimize his hash-chain to win the most blocks. Furthermore, as the seed S_j is updated for each block j, an attacker cannot look further than one block, which is the same behaviour as in proof-of-work.