Dev Document - POW In tokens

Page last modified – 12/22/21

Intro –

In this document we will go over how POW in tokens works and will go in detail about the specifications of the solidity file. Please also note for any token creators out there who would like to use our dev documents and work we would like you to credit us thank you!

Concept –

Now do not think as POW like omg its non-energy efficient because you need to understand theirs no real mining being involved. Yes, there’s a bunch of solidity contract calls but no real computation being used. This makes Stelex Lunar the most energy efficient currency just by only existing on a smart contract. Now some people may look at POW in tokens as a hoax, and sure but understand how it works first. When a user buys the token their contributing their transaction txn hash to the solidity file. This then allows them to automatically mine the currency. How it works is the owner’s wallet – automatically sends {currency} to the miner’s wallets based on network difficulty. These rewards are based on era’s like bitcoin halves, each era the block reward changes, so one day you might get 40 tokens then all of a suddenly it will change to 0.01 depending on the amount of miners.

How difficulty/mining works –

Difficulty readjusts based off how long 1 bsc block takes on avg (3s) so if theirs 1,200 an hour then we need each miner to spend 10 minutes mining each block around 200 bsc blocks total. Each time they go over the % we readjust the difficulty to meet demand thus making it harder to mine just 1 block. Every currency has this difficulty readjustment to help offset the demand to supply. Difficulty isn’t much to worry about since anybody can understand that when a consumer mines the product the product always adjusts to meet the demand to offset their supply cap. Solidity mining is simply minting – halving – and as a challenge finding out what the latest block hash is to gain a reward. Read the code below to see how the difficulty adjusts every block.

Code –

function \_reAdjustDifficulty() internal {

uint bscBlocksSinceLastDifficultyPeriod = block.number - latestDifficultyPeriodStarted;

// One BSC block = 3 sec. --> assuming 1200 BSC blocks per hour

// we want miners to spend 10 minutes to mine each 'block', about 200 BSC blocks = one Stelx epoch

uint epochsMined = \_BLOCKS\_PER\_READJUSTMENT;

uint targetBscBlocksPerDiffPeriod = epochsMined \* 200; //should be 200 times slower than BSC

//if, hypotetically, there were less BSC blocks passed in time than expected

if( bscBlocksSinceLastDifficultyPeriod < targetBscBlocksPerDiffPeriod )

{

uint excess\_block\_pct = (targetBscBlocksPerDiffPeriod.mul(100)).div( bscBlocksSinceLastDifficultyPeriod );

uint excess\_block\_pct\_extra = excess\_block\_pct.sub(100).limitLessThan(1000);

// If there were 5% more blocks mined than expected then this is 5. If there were 100% more blocks mined than expected then this is 100.

//make it harder

miningTarget = miningTarget.sub(miningTarget.div(2000).mul(excess\_block\_pct\_extra));

//by up to 50 %

}else{

uint shortage\_block\_pct = (bscBlocksSinceLastDifficultyPeriod.mul(100)).div( targetBscBlocksPerDiffPeriod );

uint shortage\_block\_pct\_extra = shortage\_block\_pct.sub(100).limitLessThan(1000); //always between 0 and 1000

//make it easier

miningTarget = miningTarget.add(miningTarget.div(2000).mul(shortage\_block\_pct\_extra)); //by up to 50 %

}

Credits

Thankyou for reading. I hope you understood what the concept of POW mining tokens are. I spent 4 hours on this paper and im glad to share with you what we have discovered so if you would like to use this as any reference in your project, please give credits to the Globe Token team thank you!.

Author –

Page written by Treyyy (CEO) Founder of GlobetokenLLC.