Slimcoin
A Peer-to-Peer Crypto-Currency with Proof-of-Burn

"Mining without Powerful Hardware"

P4Titan
(p4titan@anche.no, slimcoin@anche.no)
www.slimcoin.org

May 17, 2014

Abstract
Slimcoin is a peer-to-peer cryptocurrency derived from PPCoin's and Bitcoin's designs. Proof-of-burn joins with proof-of-stake and proof-of-work to provide block generation and network security. Proof-of-work is used as a mean for generating the initial money supply. As time passes and as the network accumulates a sufficient supply of coins, proof-of-work mining will become less necessary. Therefore, the network will rely more on proof-of-burn and proof-of-stake, the more energy efficient alternatives. Proof-of-burn is based on the idea of burning coins and generating subsequent burn hashes through a method exclusive to burn transactions. The combination of proof-of-burn, proof-of-stake, and proof-of-work strengthens the blockchain's security.

Introduction

Proof-of-work has been the predominant design for the block generation of many cryptocurrencies. By August 2012, PPCoin was released with a proof-of-stake design. Regardless, proof-of-work has maintained its dominance as the main method of generating coins. Proof-of-work is a brilliant method of distribution of the initial coins. It does, however, have issues outside the blockcahin. Proof-of-work mining requires large amounts of resources to setup, operate, and maintain. With the prices of SHA256 ASIC miners (Bitcoin and Bitcoin-like miners) increasing to unaffordable prices and graphic cards for Scrypt mining (Litecoin and Litecoin-like mining) becoming less and less available, the ability for entry into mining becomes less and less possible. In order to attain this hardware, users often have to queue into long pre-order lines. A great deal of uncertainty arises when participating in such pre-orders due to their respective drawbacks. Also, a high demand for mining hardware, causes it to be back-ordered by the distributor, wasting time that could have been spent mining.

Proof-of-Burn

Proof-of-burn is a solution to the drawbacks of proof-of-work mining. Proof-of-burn utilizes the idea of burning coins to reduce the need for powerful computational resources when mining. Proof-of-burn solves proof-of-work's dependency of powerful hardware. To acquire more mining power, instead of waiting days/months on end, all that is needed is for some coins to be burned. This idea has been implemented into the coin generation process of the Slimcoin cryptocurrency. Like PPCoin's design, this design attempts to demonstrate the viability of cryptocurrencies that can still be minted once a sufficient amount of coins have been generated, while minimally requiring powerful hardware.

Burning Coins

The role of burning coins is to be proof when mining by proof-of-burn. The concept simply involves sending coins to a burn address. A burn address is a special address that is specific and predetermined. No one has ownership of the burn address, meaning once coins are sent to it, they are forever gone and are considered "burnt". It is impossible for burnt coins to be "un-burnt". Burning coins can be parallel to buying hardware for proof-of-work mining. It costs money (burning Slimcoins), but after they are burned, they can be used to generate (mine) blocks.

Burn Transactions

When burning coins, a transaction is made to the burn address. That transaction's hash (its unique identification) is recorded separately from the other transactions' as it is a transaction that now contains burnt coins. Once a burn transaction has been recorded, so-called burn hashes can be calculated. The burn hashes are what parallel proof-of-burn to proof-of-work. Like regular transactions, burn transactions exhibit a maturating period where they have to have some depth in the blockchain (confirmations) before they can be able to calculate burn hashes.

Burn Hashes

In principal, when mining proof-of-work with physical hardware, out of all the hashes calculated over a time span 'x', only the best hash is the most important. Every other hash can be disregarded. Proof-of-burn parallels proof-of-work in that exact sense. The burn hash represents the best hash over a specific time span. In proof-of-burn, that time span is the amount of time from the creation of one proof-of-work block to the creation of the next proof-of-work block. Because of this concept, it can be said that “burnt coins are mining rigs”, the main difference being the hardware demands. In proof-of-work, the best hash is found by calculating thousands/millions of hashes and picking the smallest one out of all of them. In proof-of-burn, only one calculation is necessary, for the burn hash is the best hash.

$$\text{Burn Hash} = \text{multiplier} * [\text{Internal Hash}]$$

Every burn transaction calculates it own burn hash. The algorithm for calculating a burn hash differs from proof-of-work's algorithm. Burn hashes are calculated by multiplying a multiplier to an internal hash. The multiplier is different for each burn transaction. It follows an exponential growth curve where the independent variable is determined by the number of proof-of-work blocks found since the creation of its corresponding burn transaction. The multiplier is also inversely proportional to the amount of coins burned. The multiplier is what causes burnt coins to “decay”. Decayed burnt coins are a representation of how much the multiplier has increased. For example, if the multiplier of a burn transaction of value 60 coins triples over the course of 1000 days, the same multiplier would be achieved if on the 1000th day, 20 coins were burned. Thus, it can be said that 40 coins have decayed from the first burn transaction.

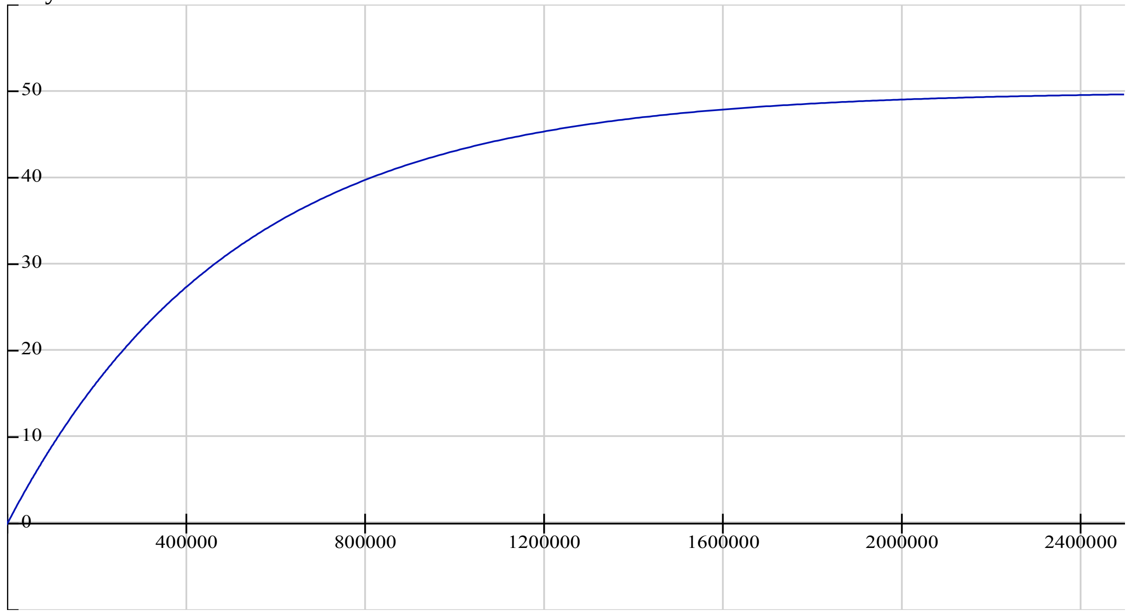

Illustration 1: From a burn transaction of 50 coins, this graph depicts the amount of coins that have decayed over the course of 240000 proof-of-work blocks.

The calculation of the internal hash is done in such a way that, if no new proof-of-work blocks have been found, it will always produce a same exact hash. That means that burn hashes have to be only calculated when a new proof-of-work block has been found. That is what makes proof-of-burn a non-intensive task, for a few hashes every minute or so can be done with a low-power computer. When being multiplied by the multiplier, the internal hash is treated as a number (256 bits). Based on how many coins have been burned and how many of them have decayed, a burn hash is calculated.

Proof-of-Burn Block Generation Specifics

There are three types of blocks in Slimcoin's design, proof-of-burn, proof-of-stake, and proof-of-work blocks. Proof-of-stake blocks were inherited from PPCoin's design and proof-of-work blocks were inherited from Bitcoin's design. To prevent proof-of-burn from becoming like proof-of-work, in the sense of non-stop mining, proof-of-burn blocks may only be found if the block preceding it is a proof-of-work block. That acts as a way to limit the amount of proof-of-burn blocks produced, since proof-of-work blocks have a time span target between blocks that is maintained by the network's difficulty. The generation of a proof-of-burn block requires the burn hash to be less than or equal to the burn hash target of the block. The burn hash target of the block is determined by using the total network's effective (mature and undecayed) burnt coins. That target is stored in the block in the form of “burn bits” and is fixed and equal for every node on the network.

Like proof-of-stake block generation, the hashing operation occurs of over a select search space (one hash per burn transaction per proof-of-work block). The hardware required to maintain that hash operation is minimal, thus the energy consumption and hardware maintenance can be considered to be near none. The block generation of proof-of-burn scales linearly, meaning if 200 coins were burned and subsequently generated 10 proof-of-burn blocks over the course of 1 day, it can be expected that if 40 coins were to be burned, 2 proof-of-burn blocks would be generated over the course of that same day, assuming the effective burnt coins in the network remained constant.

In Slimcoin's design, the hash targets for proof-of-burn, proof-of-stake, and proof-of-work are all adjusted continuously, rather than over a specific adjustment interval. That allows for better adaptability to sudden increases in network fluctuations.

Proof-of-Burn Block Verification

A proof-of-burn block must include the coordinates of a burn transaction of which its respective burn hash meets the hash target requirements. These coordinates point to exactly where the burn transaction is found in the blockchain. They contain information regarding which block and where within that block the burn transaction is found. The coordinates and are used to look up the burn transaction during the block's verification process. The signature of the block must match the signature of the in-transaction of the burn transaction pointed to by the block. That prevents the same proof-of-burn block to be copied, have its signature changed, and used again an attacker. In other words, it prevents an attacker from using someone else's burn transaction to mine proof-of-burn blocks and then have the coins generated be issued to himself instead of to the proper owner of the burn transaction.

In addition, every block in the blockchain, regardless of which type it is, has a check for the network's effective burnt coins. That value is passed down from block to block and adjusts based on the amount of newly burnt coins (its value increases) and the amount of coins that have decayed in the network (its value decreases). This value must be precise since it is used to calculate the hash target for the proof-of-burn blocks, thus it must be checked as part of every block's verification process.

Duplicate Proof-of-Burn Protocol

A duplicate proof-of-burn block detection protocol was created to prevent an attacker from using a valid proof-of-burn block to generate a multitude of copies which could be used to flood the network as a denial-of-service (DoS) attack. Every proof-of-burn block contains a value (256 bit integer) that is set to the block's burn hash. Each node collects that value of all proof-of-burn blocks that have been seen. If that value containing the burn hash differs from the calculated burn hash, that would cause its respective proof-of-burn block to be rejected (deemed invalid). If a block with the same burn hash is received, it is ignored, unless it is required by a successor block that has been orphaned (shunted away until its depending blocks are received).

The Dcrypt Algorithm

The Dcrypt algorithm is a “front-end” to the SHA256 hashing algorithm. Its main purpose is to prevent an ASIC dominated proof-of-work mining scheme from occurring in Slimcoin. Allowing Slimcoin to become ASIC minable will overshadow the proof-of-burn and proof-of-stake aspects of the coin. Dcrypt is made to be difficult to parallelize, require significant amounts of memory, require significant amounts of reading/writing to/from memory. Also, the amount of memory required is not a predetermined size. The hashing algorithm loops until a specific condition is met and for each loop, the size increases.

How the algorithm begins by hashing the data with SHA256 and creating a temporary buffer of size 65 bytes initially full of ones. Then it takes the hashed data and does a leap-frog-hash process. The first value of the hashed data is the index of value in the hashed data to append to the end of the temporary buffer. For example, if the hashed data is “0x423A8B...”, the first value is '4', so the data to append to the temporary buffer will be four from the beginning, 'A'. That temporary buffer is hashed by SHA256 and then copied into a dynamic buffer. The temporary buffer is not cleared, thus its contents still remain. This process loops again, only this time, the next value to append will be 'A' (10 in decimal) elements from that 'A'. That value is then append to the end of the uncleared temporary buffer, which then is hashed, and pushed back to the dynamic buffer. If the index of the next element exceeds the boundaries of the hashed data, a module of the size of the hashed data is applied to it and the hashed data is re-scrambled by SHA256. This entire process of hashing, appending, and pushing back, continues until the index of the hashed data is at the last value and the value there is the same as the last value of the hashed temporary buffer. Once this condition is satisfied, the original data (before any hashing) is pushed back to the dynamic buffer and this entire dynamic buffer is hashed by SHA256. The resulting hash is the hash of the Dcrypt algorithm.

A working implementation of the Dcrypt Algorithm written in Python:

```
1.    import hashlib
2.
3.    #internal hash by sha256, get the hexdigest
4.    Hash = lambda x: hashlib.sha256(x).hexdigest()
5.    SHA256_LEN = 64
6.
7.    toStr = lambda x: "".join(x)
8.
9.    def mix_hashed_nums(hashedData):
10.       #start off the tmp_list with a list full of 0xff chars
11.       tmp_list = ['\xff' for i in range(SHA256_LEN)]
12.       ret_list = []
13.
14.       hashed_end = False
15.       index = 0
16.
17.       #loop until the specific condition is satisfied, see line 37
18.       while(hashed_end == False):
19.           #attain the value of the individual character in the hashedData
20.           # adding 1 keeps a value of 0 in the hashedData moving on
21.           # increment the index by that value i
22.           i = int(hashedData[index], 16) + 1
23.           index += i
24.
25.           #if index becomes too big, mod it and re-scramble the input data
26.           if(index >= SHA256_LEN):
27.               index = index % SHA256_LEN
28.               hashedData = Hash(hashedData) #rehash
29.
30.           #get the value at the newly incremented index
31.           # push it back to the tmp_list and hash that list as a string
32.           # saving the result back into itself as a list again
33.           tmp_val = hashedData[index]
34.           tmp_list.append(tmp_val)
35.           tmp_list = list(Hash(toStr(tmp_list)))
36.
37.           #if the index is at the last character of the tmp_list
38.           # and the value there is equal to the value of the last
39.           # element in tmp_list before it was hashed (tmp_val),
40.           # the condition is satisfied, finish the loop
41.           if(index == SHA256_LEN - 1):
42.               if(tmp_val == tmp_list[SHA256_LEN - 1]):
43.                   hashed_end = True
44.
45.           #append this tmp_list hash to the large ret_list
46.           ret_list.append(toStr(tmp_list))
47.
48.       #once the condition was satisfied, return ret_list as one big string
49.       # ret_list will contain every intermediate tmp_list hash
50.       # that was calculated until the condition was satisfied
51.       return toStr(ret_list)
52.
53.   def dcrypt(data):
54.       #initially hash the data, and mix it up
55.       # then hash the mixed up data with the original data
56.       hashedData = Hash(data)
57.       return Hash(mix_hashed_nums(hashedData) + data)
58.
59.   if __name__ == '__main__':
60.       print dcrypt("The quick brown fox jumps over the lazy dog")
61.       #a74369ea2f6434aa55e38820d35300ba6130d82e0121ef64de6218c9198a377d
62.
63.       print dcrypt("The quick brown fox jumps over the lazy dog.")
64.       #559c1114f4e9de3bb12e0d1681258503c929f84527936b9d83f7c4d32a4fa67c
65.
```

Other Considerations

The Dcrypt algorithm is very intensive due to the multitude of SHA256 hashes it processes to produce one Dcrypt hash. In order to keep the propagation latency of the network as low as possible, the Dcrypt algorithm is used only to hash the block header. Transaction hashes, Merkle Trees hashes, proof-of-burn hashes, proof-of-stake hashes, etc. all are hashed by dual-SHA256. Dual-SHA256 is simply hashing the data by SHA256 and then hashing the resulting previous hash by SHA256 again. Effectively, proof-of-work is the only process that uses the Dcrypt algorithm to hash.

The calculation of the network's effective burnt coins involves division by a floating point and then a rounding down (casting to a 64-bit integer). The division by a floating point will lose precision due to the physical hardware limitations. The casting is done so the other calculations involving the effective burnt coins can be more accurate. In order to compensate for this inaccuracy, the decay factor is modified slightly pushing the deviation of accuracy to near 0%.

Conclusion

Slimcoin provides a solution for the issues that arise with hardware availability and production when regarding proof-of-work mining. At the same time, it reduces the maintenance required to set up and operate such hardware. Proof-of-burn adds an entire new layer to mining; by making mining more feasible, more people are able to participate in mining, resulting in a stronger and better network.

Acknowledgment

We greatly values the work of the Bitcoin and PPCoin developers, for their contributions to the cryptocurrency community inspired the creation of Slimcoin. On the behalf of Slimcoin, we would like to express our deepest gratitude to them.

References

Nakamoto S. (2008): Bitcoin: A peer-to-peer electronic cash system. (white paper)
(http://www.bitcoin.org/bitcoin.pdf)

King S. and Nadal S. (2012): PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake (white paper)
(http://www.peercoin.net/assets/paper/peercoin-paper.pdf)