MEMCOIN$_2$: A HYBRID PROOF-OF-WORK, PROOF-OF-STAKE CRYPTO-CURRENCY

Adam Mackenzie[1]

Abstract
Crypto-currencies are gaining traction as monetary instruments. A novel crypto-currency with a hybrid proof-of-work and proof-of-stake system is proposed that affords eventual democratic control of the monetary supply to the userbase through participatory voting. Among the features included to achieve this control are a novel a 'polymorphic' hash tree and extensive use of sequential 'memory-hard' secure hash algorithms.

Keywords
proof-of-work — proof-of-stake — crypto-currency

[1] *Please note that this white paper is a work-in-progress.*

Contents

Executive Summary

The MC$_2$ Contribution

Memcoin$_2$ (MC$_2$) offers several unique features that make it an attractive alternative to other crypto-currencies, and an important mechanism to research through scientific inquiry the various economic, social, and technical behaviours that result from participation in the crypto-currency system. These features include,

1. The use of a proof-of-work (PoW) algorithm for which an FPGA and ASIC implementation may not be easily created (*Section i*).
2. A proof-of-stake (PoS) system that works alongside the PoW system to further secure the blockchain (*Section ii*).
3. An internal participatory voting system for major controversies that may arise about the future of the blockchain, including interest rates (*Section ii*).
4. A new distribution scheme and difficulty retargeting algorithm that should afford stability to the cryptocurrency's equivalent fiat value (*Section iii*).
5. A colored coin system to allow users of the blockchain to create and maintain their own derivatives within the blockchain itself (*Section v*).
6. An embedded implementation of a lightweight client that maintains consensus of the network that eliminates the need to download the entire blockchain (*Section vi*).

The btcd Platform

MC$_2$ will be developed in btcd - an alternative full-node implementation of the Bitcoin (BTC) protocol written in Go. An alternative to bitcoind serves to improve the diversity and resilience of the crypto-currency ecosystem and its broader infrastructure. From a technical perspective, btcd has the following advantages,

1. btcd has an integrated test infrastructure, platform independent code, simpler parallelism and excellent support for concurrency, no native memory management (eliminating security vulnerabilities like buffer overflows), built-in profiling and documentation facilities, and significantly faster compilation times.

2. btcd offers a unique architecture that separates the blockchain from wallet services.

3. btcd is a clean refactor of the entire BTC protocol rather than one that is monolithic.

4. the btcd codebase strives to provide easy to follow and well-commented code.

5. btcd core packages provide more extensive unit test coverage to help prevent regressions.

Introduction

The digital currency system BTC has, in five years time, become a vast global store of wealth [4]. Similar crypto-currencies, such as Litecoin (LTC), have utilized a sequential memory-hard secure hash algorithm to make proof-of-work (PoW) mining of the chain less efficient for field programmable gate array (FPGA) devices and application-specific integrated circuits (ASIC) [5]. Recently, a BTC derived crypto-currency called Peer-to-Peer Coin (PPC) was released that introduced a novel form of block generation known as proof-of-stake (PoS) [3]. The principles of both LTC and PPC are extended in a crypto-currency system detailed below, that we refer to as MC$_2$.

i. Polymorphic, Sequential Memory-hard Secure Hash Trees for PoW

A novel approach to the PoW algorithm for MC$_2$ is the use of a polymorphic, sequential memory-hard secure hash tree. BTC began with the use of SHA256 as a secure hash algorithm, followed by sCrypt (Salsa20/SHA2-256; $N = 1024$, $p = 1$, $r = 1$) for LTC. MC$_2$ extends this approach by the use of a sCrypt utilizing polymorphic hash tree, defined as a hash tree that incorporates numerous cryptographic algorithms in a pseudo-random order. This serves to enhance FPGA/ASIC resistance as well as fault tolerance. Specifically, BLAKE512, SKEIN512, SHA3-512 (KECCAK512), and SHA2-512 are incorporated with both Salsa20 and Chacha20 stream ciphers.

The question of pseudo-random selection of these algorithms must now be addressed. To achieve this, MC$_2$ uses the Bailey-Borwein-Plouffe formula to calculate hexadecimal digits of π with a depth based on the block height of the chain (See *Appendix A*). As π is assumed to be a normal number in base 16, this should provide a suitable pseudo-random, infinite chain of numbers against which to base algorithm ordering.

An FPGA or ASIC implementation of a miner for this hashing algorithm should ideally include either an instruction cache (to contain instructions for the different hash functions of each block) or all the hashing algorithms hard-coded as logic circuits. It should be noted that this algorithm will not solve the problem of the ability to mine with FPGAs or ASICs and only delays their adoption.

ii. A Novel PoS System that Promotes Value and Prevents 51% Attacks

The PoS system in MC$_2$ is dissimilar to that of PPC and does not involve solving of stake blocks, but rather stake sub-blocks. Stake sub-blocks extend PoW blocks, confirm or reject their validity, and give PoS miners a reward for providing this service. This provides a framework for the consensus of transactions in the network among both PoW miners and stakeholders.

The first issue to resolve is how to select the PoS signatories for each block. Signatory selection should be random to prevent attacks on the network by stakeholders. MC$_2$ solves this by using data from the block header hashes of previous PoW blocks to select stakeholder tickets (See *Appendix B*). A stakeholder obtains a ticket by submitting a stakeholder submission transaction to the network with a number of coins at or above the stakeholder difficulty (some minimum quantity of coins). After submission, these coins become 'unspendable'. Over the next 23 days (16,384 blocks), PoW blocks are mined and data from their block header hashes are used to generate a random ticket from 1 to 2^{16}. After 23 days, these tickets mature and may now be used to generate stake sub-blocks.

These tickets are spent by the stakeholder when the lottery winner selected by the current PoW block is the same as the ticket number. Lottery winner generation uses a different selection algorithm compared to ticket generation that derives from several of the most recent blocks (See *Appendix B*), but should also be difficult to manipulate.

Testing will be required to demonstrate that the ticket and lottery winners generated are both unique and pseudo-random. Given that both these conditions are satisfied, the stakeholder will be able to use his or her ticket to sign a block within an average time of approximately 91 coin days after the maturation of their stake submission transaction. To prevent accumulation of stake tickets, if the stakeholder has a ticket corresponding to the lottery winner and does not vote, his or her ticket will be destroyed and the coins used to purchase them returned.

At the introduction of any new block in the network, it will instantly be known how many stakeholders are available and eligible to sign this block. As soon as the block reaches a majority or more of 'Yea' votes from stakeholders, it is accepted and locked into the block chain (See *Appendix C*). Blocks with majority 'Nay' votes are also locked in, but their transactions (including the PoW coinbase transaction) are invalidated. Blocks with less than 50% of the potential signatories reporting are orphaned. If two blocks are solved at approximately the same time, the one with more 'Yea' stake signatories is chosen as the valid chain upon the solvation of the next block. Because stake transactions are almost in-

stantly seen on the network, miners will likely begin mining the chain with more signatures visible and themselves propagate the chain with more stake transactions.

Because the valid chain is now considered to be the chain that is both the longest and has the most stake signatories, there are three things required to double spend,

1. the attacker has 51% control of the hashing power,
2. the attacker has 51% of the stake of the coin, and
3. the attacker must withhold use of their stake transactions in addition to mining PoW blocks in secret.

It be may be wise for clients to simply reject reorganizations of three or more blocks to prevent double spends, so long as there are at least a few stake signatories present on the most recently mined blocks. It should also be noted that a double spend can theoretically be performed with less than 51% of stake tickets being controlled by the attacker, but requires the attacker to mine with 51% or more of the network's hashing power for much longer periods of time, given the random probability of the lottery system for stake sub-block voting.

Given the reduced likelihood of double spend that should be afforded by using stakeholders to sign new blocks into the network, it is likely that the transactions can be considered secure after 3 blocks have entered the network and have been signed 'Yea' by a majority of the total possible stakeholders. This will need to be verified in functional implementations.

A malicious entity with 51% of stake can also perform a number of new attacks on the network. The foremost attack would be to continually invalidate blocks, stopping all network transaction traffic. This seems unlikely, as an unusable crypto-currency would rapidly decline in value. The attacker would also be able to select for which blocks are valid and which are invalid; in collusion with a large miner, this could allow the large miner to accumulate more coins which are fed back to the 51% stakeholder to solidify their cartel. This is a likely attack. Defence against this attack will likely involve the network temporarily disabling PoS signing at a previous checkpoint in favour of a pure PoW system or destroying all present PoS tickets and allowing tickets to only be generated using coins from more current blocks. Thus, a 51% stakeholder is now a more dangerous entity to the health of the blockchain than a 51% miner and the network must be vigilant to ensure that such a massive quantity of stake does not become concentrated.

There should be a means to discourage negligent stakeholder voting: for instance, continually voting 'Yea' while ignoring the contents of the PoW block. To address this problem, stakeholders not voting in the majority will be penalized by having their stake reward destroyed upon solving of the block in which their stake signature transaction is found. It should also be noted that even if the reward is destroyed, the stakeholder's original coins are still returned to them upon consumption of the ticket.

The last problem addresses the possibility of a blockchain fork arising from two blocks entering the network at exactly the same time and being signed by exactly the same number of signatories with exactly the same number of potential signatories. If two blocks have the same number of signatories but one has a lesser number of potential signatories (based on the lottery winner), we simply select this block. However, in the unlikely event that both the number of signatories are the same and the number of potential signatories are the same, the network simply selects the block with the larger amount of work present in the block header hash as being correct (See *Appendix C*).

iii. Miner Reward Algorithms and the Rate of Block and Coin Generation

The base rates for the MC_2 network differ from those in previous crypto-currencies in order to simplify the system: an MC_2 year is 360 days (a coin year), difficulty adjustment periods are every 4.5 days, PoW and PoS block reward adjustment periods are every 9 days, and a PoS stakeholder transaction maturation period is approximately 23 days (16,384 blocks) (See *Appendix D*). The network target of MC_2 is 30 blocks per hour for PoW and 150 new stake submission transactions per hour for PoS.

The rate of progressively reduced block reward and constant block generation in BTC PoW block generation results in massive disinflation[1] as network time progresses. This is expected to promote the adoption and usage of BTC as a monetary instrument. By contrast, PPC uses a PoW block reward algorithm that can both inflate and disinflate depending on the network hash rate and makes the assumption that network hash rate will be consistently increased exponentially. In addition, PoS blocks in PPC apply a 1% per annum interest to stake miners who obtain them. MC_2 abandons PPC's PoS reward generation algorithm and employs a similar system of block reward seen in BTC and LTC for PoW blocks, with faster block reward adjustments. The average quantity of stake reward is calculated similarly. This provides a strong incentive for the mining of both PoW blocks and block confirmation by PoS stakeholders.

Reward adjustments are performed every 9 days resulting in an 8% decrease in value per 360 day MC_2 annum (8% disinflation rate) starting with a base PoW reward of 250 coins per block and a base PoS reward of 125 coins per block

[1] Please note that inflation in this paper refers to the increase in supply of a given crypto-currency, while disinflation refers to the decrease in supply inflation of a crypto-currency.

by the equation (See *Appendix D*),

$$R_{current} = R_{initial}\left(1 - \frac{0.08}{40}\right)^{\lfloor \frac{block\ height}{6480} \rfloor}. \quad (1)$$

A lower starting reward for stake is given to ensure that the incentive for PoW miners is always greater than the incentive for PoS miners, as the former perform more computational work. Given a target rate of 30 blocks $h-1$, the chain should approach a daily amortized inflation rate of approximately 1% (See *Figure 1*) as it reaches coin year 27 or block 6,998,400. At this point, the interest rate stemming for both PoW and PoS rewards becomes locked in for one coin year and is then controlled by the user-base rather than being hard-coded into the chain itself. 'Yea' or 'Nay' votes are given with each block obtained in the block header; at the end of each coin year from block 6,998,400 onwards, the sum of all votes for both PoW and PoS are tallied as 'Yea' (1) or 'Nay' (-1), summed, and divided by the total number of votes (blocks for this coin year) to yield an elected fraction v. This fraction is then the difference in inflation applied to each work system for the following coin year, with a maximum difference of 1% total.

A requirement for the PoS inflation percentage is that it may not democratically elect a PoS reward to a value of more than 50% of the PoW reward by inflation, to prevent older PoS signatories from overtaking the chain and affording control of the network to a group of hoarders who perform virtually no work (a problem observed with current global financial systems). A requirement for both the PoW and PoS inflation rates is that they do not go below 0%, as this removes incentive to mine or sign PoW blocks.

We can represent the overall inflation of chains mathematically as an amortized daily inflation rate per time period with function (2), where the amortized daily inflation rate f_i is defined by the time period T (365 days in *Figure 1*), the coins issued per day b, and the total number of coins in existence b_T,

$$f_i = \frac{1}{T}\sum_{i=1}^{T}\frac{b_i}{b_T}. \quad (2)$$

By this metric, we can observe that the rate of disinflation for MC_2 is less than that of BTC and LTC five years after conception. Dissimilar to PPC's constant PoS inflation rate, the PoS rate of inflation in MC_2 is precisely the same as that of the PoW rate until reaching coin year 27, at which time the rate is frozen for one coin year and then handed over to the democratic control of the miners.

The difficulty for MC_2's PoW chain is calculated from the linear weighted moving average of block times over the past 18 days (See *Appendix D*). The stakeholder difficulty is likewise calculated from the linear weighted moving average of new stake submission transactions per block for the past 18 days.

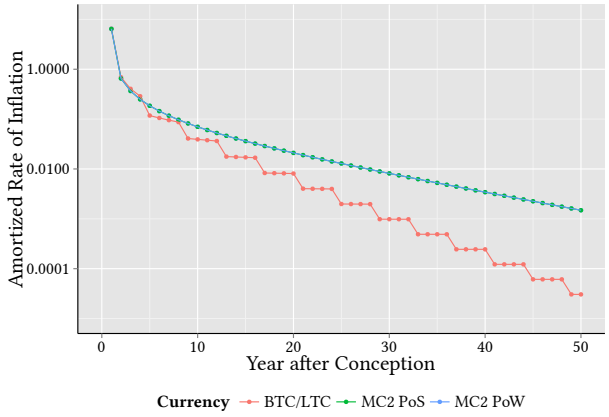

Fig. 1. Amortized rates of inflation for various crypto-currencies for a given year (365 days). Note that the amortized rate of inflation represents a yearly inflation rate calculated from daily rates of inflation (see *Equation 2*).

iv. Modifications to the Block Header and Transaction Data

The following extra data are appended to the PoW block header as compared with a BTC block header (See *Appendix E*),

1. Current stakeholder difficulty (64 bits).
2. Which of the 8 sCrypt algorithms is used for a PoW block (3 bits).
3. Five vote bits (2 for PoW and PoS reward adjustment, 3 unused) (5 bits).
4. Lightweight whole chain ledger SHA3-256 hash (256 bits).
5. The SHA3-256 bit hash of the sorted list of stakeholder sub-block hashes (alphabetically by block header hash), used to sign the previous block into the network (256 bits).

The second component is necessary for clients who are downloading and verifying the blockchain to be able to easily do so.

Vote bits only affect the block during democratic voting cycles for the rate of inflation (See *Section iii*). Remaining vote bits may be used to decide other issues affecting the chain as time goes on.

Transactions will largely stay the same as in BTC, save for colored coin transactions (See *Section v*).

Stake submission transactions are similar to coinbase transactions except they are required to have inputs (up to 64) (See *Appendix B*). These inputs must follow the rules specified in *Section ii.* The exact reward claimed in the coinbase will be

easy to determine and verify based on the reward algorithm from *Section iii.*

Stake sub-blocks contain the following (See *Appendix C*),

1. PoS block header hash (256 bits, SHA3-256, difficulty 2).
2. PoW recent block header hash (512 bits).
3. Single bit voting 'Yea' (1) or 'Nay' (0) on the new block (1 bit).
4. Five other vote bits (two for PoW and PoS reward adjustment, three unused).
5. Ledger hash.
6. A transaction list containing only the stakebase transaction signed by the stakeholder address delivering the reward to the ticket holder. This transaction verifies that the block is being sent from the stakeholder.

An additional modification to MC_2 is the automatic sorting of the transaction ledger (See *Appendix F*) by coinbase transaction, then by stake submission transactions, then by fees, then by address. By sorting the hash tree of transactions in this way, we prioritize inclusion of transactions into the blockchain based upon fees. Because of the PoS voting system in place, it is ensured that persons submitting transactions to the network with a large enough fee will be able to incorporate their transactions in the blockchain and PoW miners can thus not selectively exclude transactions as in BTC.

MC_2 alleviates BTC's "Red Balloons" problem [1] by destroying fees for the first 32,768 blocks and then transferring fees from the current block to the coinbase address found in the block 32,768 blocks prior (See *Appendix G*).

v. Colored Coins

The introduction of arbitrarily generated "colored" coins that map to an originator address can serve useful purposes (See *Appendix H*). The first is so that any organization can generate their own coins easily and trade them with clients over the network. Once example would be an organization that wishes to reward users for solving a problem of biological significance such as finding the minimum energy structure for a given protein. The organization could use their own centralized network to determine the reward for their client, then transmit the equivalent number of colored coins to them over the network. Note that the organization would still be required to pay the fees associated with the network in network coins.

The second useful purpose is for decentralized exchange using the network as an escrow. This is performed using a unique transaction script called a conditional transaction (See *Appendix H*). Exchange would use two conditional transactions: one for a colored coin, and one for another colored coin or for network coins. The transactions would send coins to their corresponding output addresses if and only if the corresponding transaction to their input is included in the same block.

Coins would be generated by a genesis transaction. After the genesis transaction, the colored coins could be spent as normal coins; however, the colored coins would have to be transmitted throughout the network with a special field known as the originator address that specifies the address of the genesis transaction output and a script flag indicating that they are colored. Tracking colored coins in the network would be optional for the client, who is able to specify which colored coins to watch by address. Usage of colored coins requires the full-weight client, as they are not included in the lightweight whole chain ledger.

vi. Lightweight Clients

One problem with BTC is the massively growing blockchain size. The solution herein involves the generation of a lightweight whole chain ledger. The lightweight whole chain ledger contains the following information,

1. The SHA3-256 hash of this ledger and the block height at which it is found.
2. All addresses containing more than 0.0001 coins, the quantity of coins in these addresses (minus coins reserved by stake submission transactions), and the block heights in which the number of coins in the address last changed. The addresses are sorted by value of the address.
3. Block header hashes used as PoW for every block.
4. All currently submitted and unredeemed stakeholder transactions, their tickets (if applicable), and whether or not they have had the opportunity for redemption (their ticket was selected but they chose not to redeem it, which determines long-term eligibility).

The list of addresses and associated block height for the last transaction is stored as a sorted B+ tree on disk for both ease of access and simplicity in the addition of new addresses. This list is hashed by both PoW miners and PoS stakeholders, and should be congruent among both, except in the potential case of a PoS 'Nay' vote.

To enhance security, the lightweight client should also hold the last 256 blocks in memory as well as the transaction lists of the previous block (which the network almost completely reverts to in the event that the block is voted against by stakeholders and which may be required in the event that some inconsistency arises). Upon the addition of a new block, the last block is in the 256 block high list pruned. In the event

of a large reorganization (unlikely), the list will need to be reacquired from one of the full-weight clients, which can recalculate and redistribute them.

References

[1] Babaioff, M., Dobzinski, S., Oren, S., and Zohar, A. On Bitcoin and Red Balloons. In Proceedings of the Association for Computing Machinery (ACM) Conference on Electronic Commerce (2012).

[2] Karame, G., Androulaki, E., and Capkun, S. Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin. In Proceedings of the Association for Computing Machinery (ACM) Conference on Computer and Communications Security (2012).

[3] Nadal, S. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. Self-published (2012).

[4] Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. Self-published (2008).

[5] Percival, C. Stronger Key Derivation via Sequential Memory-Hard Functions. Self-published (2009).

Appendix A: PoW Hashing Function

For sCrypt, we know that there is a linear relationship between the size of N and the time required to perform a hash from the data compiled by WindMaster on a CPU using the original complete LUT algorithm with no TMTO.

We will assume that this is also true for any given GPU. We can then assume that for any given PoW difficulty ($REAL\,DIFFICULTY$) at any arbitrary N value, we can scale the difficulty in a linear fashion such that block times for any N will be consistent. We do this in the following way,

$$CURRENT\ BLOCK\ DIFFICULTY = \frac{SMALLEST\ POSSIBLE\ N}{CURRENT\ N} REAL\,DIFFICULTY. \tag{3}$$

However, we know from mtrlt's data that when using algorithms employing a lookup gap/TMTO, that this relationship does not necessarily hold true and becomes unpredictable.

For the sake of ease of implementation and detachment from vulnerabilities relating to variable scaling of performance with N factor changes, MC_2 (white paper 0.04+) will now employ a fixed value for N, with $N = 8192$.

The performance of ChaCha20 and Salsa20 with various PBKDF2 implementations appears to be close enough that we can continue to use them interchangeably and rotatably as stated in previous versions of the paper. However, instead of using the blockchain as the source of inbuilt entropy, let us rather use the hexadecimal digits of π.

We will begin with the following sorted order of hashes, with their bins given on the left in brackets,

Bin	Sorted Order	Hash
{1}	000-031	Salsa20/BLAKE512
{2}	032-063	Chacha20/BLAKE512
{3}	064-095	Salsa20/SKEIN512
{4}	096-127	Chacha20/SKEIN512
{5}	128-159	Salsa20/SHA3-512
{6}	160-191	Chacha20/SHA3-512
{7}	192-223	Salsa20/SHA2-512
{8}	224-255	Chacha20/SHA2-512

Tab. 1. Sorted Order of Hashes with Bins

Now, for any given cycle of blocks ($Cycle\ Size = 8$) we need to establish a random order for these hashes. We will do so by first calculating the hexadecimal digits of π as follows,

$$\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6}, \tag{4}$$

which returns the decimal value of the 4-bit hexadecimal number at position i of π. It is generally held that π is a normal number in base 16 (and 2), and we can assume that the distribution of bits in base 2 will be completely random.

We will calculate a total of 16 hexadecimal numbers (128 bits), starting at $(2\,(BLOCK\ HEIGHT\ FOR\ CYCLE\ START))$ and ending at $(2\,(BLOCK\ HEIGHT\ FOR\ CYCLE\ START) + 16)$.

Let's say we begin at block 8,192, or cycle 1,024. We calculate the 16,384$^{\text{th}}$ - 16,400$^{\text{th}}$ digits of π by the formula above to yield the following (not actual digits of π; an example),

50 *A*2 0*E* 33 *F*7 6*B* 10 *E*9.

This is the following list converted into decimal one byte at a time: 80, 162, 14, 51, 247, 107, 16, 233. Now we can begin assigning these to bins: 80 = **{3}**, 162 = **{6}**, 14 = **{1}**, 51 = **{2}**, 247 = **{8}**, 107 = **{4}** ...

We hit a problem upon reaching 16, because we have already used the bin of **{1}** for 14. The MC_2 specifications say that we may only use any given hash function once per cycle. So, we simply overflow to the next bins sequentially until we find one that hasn't been used before. The first overflow is to **{2}**, then **{3}**, etc. Bins **{2}**, **{3}**, and **{4}** have already been used, but bin **{5}** has not. So, in this instance 16 = **{5}**. Similarly, 233 refers to bin **{8}**, which is also already used. The only bin not used up to now is **{7}**, so in this instance 233 = **{7}**.

Our final list of hash functions for blocks 8,192-99 is as follows: **{3}**, **{6}**, **{1}**, **{2}**, **{8}**, **{4}**, **{5}**, **{7}**.

Appendix B: Proof-of-Stake Voting for Proof-of-Work Blocks

Network Input: Proof of Stake Enrolment, Stakeholder Submissions, and Difficulty

STAKEHOLDER DIFFICULTY The number of coins required to successfully submit a stake transaction to the network. Adjusted to meet a target of 150 new stake submission transactions per hour, enough so that on average there will be 5 stakeholders verifying each PoW block. The stakeholder difficulty is stored as a 64-bit integer.

LOTTERY WINNER The lottery winner is calculated from the concatenation of 10 bits from 10 of the last 20 blocks' block header hashes and six bits from the current block's block header hash.

An example is below:

BLOCK_HEADER_HASH(BLOCK00000): **1**0111010101000110...1010101001110101 (single bit used)
BLOCK_HEADER_HASH(BLOCK00002): 10101000010100100...1101101100010100 (single bit used)
... (position of bit selection incrementing 16 bits every 2$^{\text{nd}}$ block until block 20, the current block)
BLOCK_HEADER_HASH(BLOCK00020): 01010000100001100...**011011**1001010101 (current block; use 6 bits)

_____ $\Rightarrow$ 10.....011011
Reverse bit string
_____ $\Rightarrow$ 110110.....01 (LOTTERY WINNER)

NOTE: It may not be immediately clear why we use 6 bits from the most recent block. Consider the attack in which a malevolent stakeholder has the majority of tickets for any given block header hash. If we use only old blocks, especially ones far back in the history of the blockchain, no matter how many new proof of work blocks are generated the attacker can still halt the chain. Now consider the instance in which 6 bits are taken from the most recent block; we have $2^6 = 64$ possibilities for the lottery winner. The malicious stakeholder in this case can block only one of 64 possible blocks from entering the blockchain; in the event this block is not voted on by the malicious stakeholder, PoW miners can continue work from the previous block and generate another block with a different lottery winner. Then, non-malicious stakeholders can push this block through the network and the chain can continue. *It is extremely important that PoW miners continue on old work until the latest block has been verified by a majority of stakeholders for this reason.* This also gives an interesting trade-off for security: we can choose up to 16 bits from the current block, however, the more bits we choose from the current block, the more readily a PoW miner can manipulate the lottery winner. Using blocks that are weeks or months old mitigates this manipulation, however, without choosing several bits from the current block we enter the case where an attack as detailed is extremely likely. So, we must select n bits from the current block at a level that allows for some options in terms of lottery hash, but which does not allow for the complete manipulation of the lottery winner. In the case of 6 bits (64 possible lottery winners), the PoW attacker mining at the current block can only generate $\frac{64}{65536}100\% = 0.01\%$ of possible tickets.

NOTE: Why not use older blocks? Ideally, we don't want an attacker to know what the lottery winner is far in advance in fear that they may accumulate tickets containing these bits. For instance, if we use all 10 bits from 14 days in the past except for the remaining 6 bits from the current block, the attacker needs only $64 \times 3 = 192$ tickets corresponding to these known bits in order to effectively halt the blockchain by not voting. Hence, we should try to use very recent blocks to select these 10 bits.

The block number (BLOCK#####) represents the block 20 blocks before the current block; that is, BLOCK00000 is the block 20 blocks prior to the current block. The BLOCK_HEADER_HASH is the last 256 bits of the block header hash for each block.

NOTE: There are many sources of entropy in the network, but we want to use the ones which are the most consequential to manipulate. In our case, it's always the block header hash as it's unlikely a PoW miner will throw away blocks it generates in hopes of manipulating the ticket system. This is because finding a proof of work block is very computationally intensive.

TICKET The ticket is calculated from the concatenation of a bit from the last 256 bits of 15 of the last 16,384 blocks' block header hashes, a single bit from the last 256 bits of the current block's block header hash, and the relative position (depth) in the transaction hash tree. In the case of the block header hashes, one bit is selected to be added to the data to form the 'root' of the ticket. To this root, we can give greater uniqueness and randomness to tickets by shifting the bits used in the 16 blocks to the right based on the depth within the transaction tree of the original stake submission transaction.

Example (for a stake submission transaction that is depth 2 in the transaction tree of the PoW block, resulting in the bits we are selecting being shifted rightwards 2 positions),

BLOCK_HEADER_HASH(BLOCK00000): 10**1**1101010100011011...1010101001110101
BLOCK_HEADER_HASH(BLOCK01024): 1010100001010010000**0**...1101101100010100
... (position of bit selection incrementing 16 bits every 1,024$^{\text{th}}$ block)
BLOCK_HEADER_HASH(BLOCK16384): 0101000010000110010...01**0**0111001010101 (current block)

_____ ⇒ 10.....0
Reverse bit string
_____ ⇒ 0.....01 (TICKET)

NOTE: Duplicate tickets can arise if more than 256 stake submission transactions are submitted to the network; it should be the case that the maximum number of stake transmission transactions for any given PoW block be limited to 256, although there are other potential workarounds. However, a limit of 256 is reasonable as the target is only 5 stake submissions per block.

The block number (BLOCK#####) represents the block in which the stakeholder submission transaction was included in the blockchain. The BLOCK_HEADER_HASH is the last 256 bits of the block header hash for each block. To these 15 bits a single bit from the block header hash of the newly submitted PoW block (the last bit) is added; yielding 16 bits total.

This yields a ticket, a 16-bit number that, when it matches the current lottery winner, may be used to obtain a stakeholder sub-block.

Overview of Stakeholder Sub-block Voting Algorithm

1. A stakeholder submits a stakeholder submission transaction to the network. This transaction sends coins from the user's wallet to a newly generated address also owned by the stakeholder. This stakeholder submission transaction is fee-free, but must meet the following requirements,

   A. The quantity of coins submitted is greater than or equal to the current stakeholder difficulty.

   B. The number of inputs is no greater than 64 unique addresses. If the number of inputs is greater, a fee will need to be paid to the network for the consumption of bits beyond this quantity.

2. The stakeholder submission transaction is incorporated into the blockchain in the next PoW block and confirmed by the PoS miners. The output address of the stakeholder submission transaction is now unspendable.

3. In 16,384 blocks (∼22 days), the stakeholder submission transaction matures and yields a 16-bit ticket.

4. From the 16,384$^{\text{th}}$ block onwards, the stakeholder waits to see if the lottery winner hash selects their 16-bit ticket. In the event they are selected, they must immediately submit their stakeholder sub-block to the network and either confirm or deny the most recent PoW block. Two possible scenarios exist from here,

   A. The stakeholder submits their voting sub-block to confirm or deny the newly minted PoW block. The stakebase transaction in this block affords their reward (the current stake reward) as described in *Appendix D*. The output address containing both the coins used to submit the stakeholder submission transaction and the reward is unlocked and is now spendable.

   B. The stakeholder fails to submit their voting sub-block before the next incoming, confirmed PoW block. In this case, their PoS ticket is invalidated, no reward is issued, and the funds in the original output address are unlocked and thus spendable by the original stakeholder.

NOTE: Destroying tickets is necessary to keep users from trying to overtake the network by hoarding PoS tickets. Implementations of PoS as described should allow users to encrypt and lock all addresses in their wallet except the stake submission transaction output addresses, as these need to be unencrypted and ready to use at any given time.

Appendix C: Proof-of-Work and Proof-of-Stake Hybrid Design

PoW/PoS Hybrid Protocol

Chain Selection

The valid chain is selected by the following formula in general,

$$CHAIN\ SCORE = \sum\left((PoW\ DIFFICULTY)^{0.5}\,(PoS\ DIFFICULTY)^{0.5}\,(NUMBER\ OF\ STAKE\ VOTE\ BLOCKS)^{0.5}\right) for\ all\ blocks. \tag{5}$$

If a dispute arises from equivalence of these terms, then we resolve the dispute as follows,

$$CHAIN\ SCORE = \sum\Big((PoW\ DIFFICULTY)^{0.5}\,(PoS\ DIFFICULTY)^{0.5}\,(NUMBER\ OF\ STAKE\ VOTE\ BLOCKS)^{0.5}\,(NUMBER\ OF\ POTENTIAL\ VOTERS)^{0.5}\Big)\ for\ all\ blocks. \tag{6}$$

If a dispute again arises from equivalence of these terms, then we resolve the dispute as follows,

$$CHAIN\ SCORE = \sum\Big((PoW\ DIFFICULTY)^{0.5}\,(PoS\ DIFFICULTY)^{0.5}\,(NUMBER\ OF\ STAKE\ VOTE\ BLOCKS)^{0.5}\,(NUMBER\ OF\ POTENTIAL\ VOTERS)^{0.5}\,(TOTAL\ WORK)^{0.5}\Big)\ for\ all\ blocks. \tag{7}$$

The last two terms are intended to resolve potential dispute arising from, respectively, identical chains in terms of,

1. Votes, PoW difficulty, and PoS difficulty.
2. Votes, PoW difficulty, PoS difficulty, Number of Potential Voters.

Total Work is defined by the sum of the number of trailing zeroes and their last digits in the block header hash; for instance, if there are two chains identical in every way but one block header begins with 0000#... and the other with 000#... , the 0000#... block containing chain is considered the winner and the other is orphaned.

An exacting method of selection is required to prevent network forks.

Overview

1. A new block containing various transactions, a block header, and a block header hash at or above the current PoW difficulty enters the network. If the there are no tickets corresponding to the lottery winner for the block, the block is automatically orphaned. Otherwise, *GOTO 2*.
2. PoS miners see the new block and verify that it contains a reasonable list of recently published transactions from the merkle root, then vote on the block by submitting a special stake block to the network, and signing it from their stake address.
3. Any given block targets an average of 5 stake voters (PoS miners/voting blocks), but may have less or more due to entropy in the chain. For any given number of voters, there are three conditions that may arise,

   A. The stake voters vote in the majority to approve the transactions in a block. The block is considered valid in its entirety and PoW mining continues from this block. Stakeholders who had voted 'Yea' (with the majority) have their rewards issued, while absent voters and 'Nay' voters lose their rewards. *GOTO 4*.

   B. The stake voters vote in the majority to reject the transactions in the block. Stakeholders who had voted 'Nay' (with the majority) have their rewards issued, while absent voters and 'Yea' voters lose their rewards. The block transaction list (including the coinbase transaction and any stake submission transactions) and ledger hashes of this PoW block are rejected, but PoW mining continues from this block header and the block headers of the stake blocks. *GOTO 4*.

   C. The majority of stake voters neglect to vote at all, and the PoW block is rejected. Ignore this block. *GOTO 1*.

   NOTE: While waiting for PoS miners to vote with blocks, PoW miners can continue mining from the previous PoW block just in case this new block ends up being rejected. Pools will have to figure out how to deal with stales from this.

   NOTE: Block header hashes for PoW must still be valid according to the network difficulty; we don't want a case where PoS miners can vote in totally invalid PoW blocks.

4. A SHA3-256 hash of the ordered (alphanumerical) block header hashes from the stake blocks of the previous round is incorporated into the block header of the new PoW block header being hashed and PoW mining continues as normal.

In the event that some of the ticket holders are selected and fail to generate vote blocks during the previous round, the original sum of coins invested into a stake submission transaction is returned to them and the ticket is considered destroyed. This doesn't have to be explicitly declared in the chain, as anyone with the block chain will be automatically able to tell if these tickets are invalid or not (because they will have missed their chance to vote on a previous block).

Notes on Possible Attacks
One potential attack involves a PoW miner with 51% of the network hashrate or more and some number of stake votes (10% or more is ideal). Any PoW miner submits a block (Block A) to the network, and a majority of the benevolent stakeholders vote while the attacker's malicious stakeholders do not. All miners on the network continue looking for the next block (Block B); eventually benevolent miners submit a block (Block B extending Block A) to the network containing a hash of the benevolent stakeholder's voting sub-blocks. The block is immediately voted on by 5 PoS voters - an assumption for simplicity's sake. In secret, the attacker mines a PoW block (alternative Block B) and then submits one or more additional PoS sub-blocks for the last block to the network, followed by the submission of his or her malicious alternative Block B to the network. The block is immediately voted on by 5 PoS voters - an assumption for simplicity's sake. The benevolent PoW miner's block (Block B) is then invalidated because the chain with the alternative Block B has more PoS votes, and the attacker in this way is able to route the reward of most PoW blocks to himself.

However, such an attack exposes the attacker to risk as well: if his or her new fork fails to get as many PoS votes as the one of the benevolent miners, he will lose the reward of his or her stake tickets. This should provide a deterrent effect, but it remains to be seen how effectively or often utilized an attack can be on the main network.

Structure of PoS Vote Sub-block
Difficulty 2 SHA3-256 block header containing the following extra fields,

1. PoW recent block header hash.
2. Single bit voting 'Yea' (1) or 'Nay' (0) on the new block.
3. Five other vote bits (two for PoW and PoS reward adjustment, three unused).
4. Ledger hash.
5. A transaction list containing only the stakebase transaction signed by the stakeholder address delivering the reward to the ticket holder. This transaction verifies that the block is being sent from the stakeholder themself.

Appendix D: Reward and Difficulty Algorithms

Starting Rewards
Proof-of-work: 250 coins per block.

Proof-of-stake: 25 coins per stake vote (or average 125 coins per block).

Network Targets and Constants
Proof-of-work: 2 blocks per minute.

Proof-of-stake: 5 stakeholder voting sub-blocks per PoW block.

Reward adjustments: Every 9 coin days (6,480 blocks).

Difficulty adjustments: Every 4.5 coin days (3,240 blocks).

Reward Adjustments
Rewards are adjusted every 9 coin days (6,480 blocks) according to the following formula,

$$REWARD\ CURRENT = REWARD\ INITIAL\left(1-\left(\frac{0.08}{40}\right)\right)^{\lfloor\frac{BLOCK\ HEIGHT}{6480}\rfloor}. \tag{8}$$

The same formula is used for both PoW and PoS blocks.

Difficulty Adjustments: PoW
Difficulty adjustments occur every 4.5 days (3,240 blocks) according to the following formula,

$$A = \frac{\sum(TIMING\ TARGET - TIME\ BETWEEN\ BLOCKS)\ for\ previous\ blocks\ in\ adjustment\ period}{ADJUSTMENT\ PERIOD}, \tag{9}$$

where *TIMING TARGET* is 2 minutes and *TIME BETWEEN BLOCKS* is the difference in time between any given two adjacent blocks in the adjustment period (*ADJUSTMENT PERIOD*; 3,240 blocks). Now we calculate the difficulty adjustment by using the linear weighted averages of A from four time periods (18 coin days),

$$D = \frac{\sum N(A_N)}{\sum N}\ for\ N = \{1,2,3,4\}, \tag{10}$$

where A_N is the value of A for difficulty period N (for $N=1$, beginning 18 coin days ago; for $N=2$, beginning 13.5 days ago; etc.). This yields the percent change in difficulty. For example, if A in period 1 (oldest) is 0.05, in period 2 is 0.10, in period 3 is -0.05, and in period 4 (latest) is 0.10,

$$D = \frac{\sum(1(0.05)+2(0.10)+3(-0.05)+4(0.10))}{10} = 0.05. \tag{11}$$

The new difficulty is now $(1+D)\,PREVIOUS\ DIFFICULTY$. For any given A, the maximum value (change) is a 400% increase (4.00), and the minimum value is a 75 decrease (0.25).

Difficulty Adjustments: PoS
Difficulty adjustments occur every 4.5 days (3,240 blocks) according to the following formula,

$$A = \frac{\sum(STAKE\ SUBMISSION\ TARGET - STAKE\ SUBMISSION\ PER\ BLOCK)\ for\ previous\ blocks\ in\ adjustment\ period}{ADJUSTMENT\ PERIOD}, \tag{12}$$

where *STAKE SUBMISSION TARGET* is 5 and *STAKE SUBMISSIONS PER BLOCK* is the number of stake submissions per any given block in the adjustment period (*ADJUSTMENT PERIOD*; 3,240 blocks). Now we calculate the difficulty adjustment by using the linear weighted averages of A from four time periods (18 coin days),

$$D = \frac{\sum N(A_N)}{\sum N} \; for \, N = \{1,2,3,4\}, \tag{13}$$

where A_N is the value of A for difficulty period N (for $N = 1$, beginning 18 coin days ago; for $N = 2$, beginning 13.5 days ago; etc.). This yields the percent change in difficulty. The new difficulty is now $(1+D)\,PREVIOUS\,DIFFICULTY$. For any given A, the maximum value (change) is a 400% increase (4.00), and the minimum value is a 75% decrease (0.25).

Appendix E: Adjustments to the Proof-of-Work Block Header

The following are appended to the block header,

1. Current stakeholder difficulty (64 bits).
2. Which of the 8 sCrypt algorithms is used for a PoW block (3 bits).
3. Five vote bits (two for PoW and PoS reward adjustment, three unused) (5 bits).
4. Lightweight whole chain ledger SHA3-256 hash (256 bits).
5. The SHA3-256 bit hash of the sorted list of stakeholder sub-block hashes (alphabetically by block header hash), used to sign the previous block into the network (256 bits).

Appendix F: Adjustments to the PoW Transaction Tree

The Proof-of-Work (PoW) transaction tree in MC_2 is sorted in the following way,

1. The coinbase transaction for the PoW miner appears as the first transaction.
2. Stake submission transactions always appear next and are sorted by their stake output addresses. The number of stake submission transactions is limited to 256 per block.
3. After the stake submission transactions, all other transactions appear sorted (in order of importance) by transaction fee amount, the first input address (signing address), and the output address (if present).

The transaction tree MUST satisfy the following condition: for any given unique list of transactions in a random order containing the same transactions, there exists one and only one sorted transaction tree containing this same list of transactions. This effectively prioritizes transactions based on fee and can also be used as a mechanism to verify consensus among clients in the network (although this latter property is not a major focus of the developers at this time).

Appendix G: Addressing the "Bitcoins and Red Balloons" Issue

One of the problems with BTC is that it incentivizes the hoarding of transactions with fees, because the PoW miner who incorporates these transactions obtains these fees. MC_2 attempts to address this simply by the following method,

1. For the first 32,768 blocks, all fees are destroyed.
2. For all blocks after the first 32,768 blocks, fees are directed to the coinbase address of the block 32,768 blocks before the current block.

The PoW miner now has less of an incentive to hoard transactions with fees, as the fees are less likely to go directly to himself.

Further, the PoS mechanism also addresses this issue. If a client submits a transaction with a fee to the network, the PoW miner may see it and attempt to hoard it by not propagating it; however, if the transaction is seen by the PoS miners who are required to validate the PoW block, the PoW miner will see his or her block transactions (including fees and block reward) invalidated by the PoS miners. Hence, hoarding transactions by PoW miners is strongly disincentivized (See *Appendix J*).

Appendix H: Colored Coin Transactions and Conditional Transactions

Aside from regular transactions (destroying coins at the originating address and generating them at the target address, multisig transactions, and so on as dictated by the normal BTC scripts), colored coins have a special type of transaction called a **conditional transaction**. Conditional transactions are allowed via new scripting components.

A conditional transaction satisfies the following,

1. Signed for by the private key of the address, which possesses a quantity of colored or regular coins that it offers in exchange for a quantity of colored or regular coins existing at another address.
2. Has a number of blocks for which the conditional transaction is valid for beyond the current and may be satisfied by a corresponding conditional transaction (0 = this block only).
3. Has both a requested coin type (either MC_2 or some other colored coin, given by the originator address and the block in which the colored coin genesis transaction occurred) and a quantity of these coins required to be possessed by the person making the converse conditional transaction.
4. Has an address (not owned by the person making the present conditional transaction) from which the coins in 3. will be assumed to be coming from.
5. Has a unique, unused output address owned by the person making the transaction for the requested coins to go to.
6. (Optional) A string up to 512 char in length indicating the e-mail address to contact the person offering the conditional transaction, etc.

After the conditional transaction enters the blockchain, it will not be considered spent until a corresponding conditional transaction signed by the address in 4. exists on the network, and if and only if this address has the requested quantity of coins desired by the original conditional transaction. For any given conditional transaction, coins will be considered unspendable for the duration of blocks for which the conditional transaction is specified to be valid, unless a corresponding conditional transaction enters the block chain and the two types of coins are exchanged using the chain as an escrow.

The intended usage of colored coins and conditional transactions is to facilitate decentralized exchange of goods. Let's say John has 100 gold coins and wishes to exchange them for MC_2. He can then create 100 destructible JohnCoins corresponding to his gold coins, and a link (via the optional string) to a pastebin containing the details of his coin, where to contact him, and a PGP signature to keep communications private - for instance, "*JohnCoins 1/4 oz gold rounds upon redemption, shipped to USA. pastebin.com/mypaste*" This text then appears in the client window with the number of JohnCoins generated. John can now send these coins to people in exchange for a quantity of MC_2 of his choosing by a conditional transaction - they just need to send him an e-mail with their address so he can generate a conditional transaction and upload it to the network. Once these JohnCoins enter circulation, they can now be traded amongst users as if they were actual gold coins. When a user decides he wants to redeem a gold coin, he just sends them to John with a normal transaction containing a message of some sort specifying where he or she wants the physical gold coins sent (e.g., a link to a pastebin containing PGP-encrypted text for John to read); because John specified the coins were destructible, any coins sent back to him are destroyed upon arrival and removed from circulation.

The benefit to such a trading mechanism is that there are no fees associated with it except those incurred to support the PoW miners on the network, and that it is nearly completely decentralized as it operates within the blockchain. Further decentralization could be achieved by using a TOR service for e-mail or announcements, etc.

Because we can tell from the blockchain who is redeeming the coins (by address), persons owning these addresses can even go so far as to give feedback about John's services through a public site by writing a message and signing it with their private key. This type of community feedback keeps users anonymous while enabling them to inform other users of whether or not a seller's goods or services are legitimate.

Another use for colored coins is rewarding online bulk computational efforts, e.g., *WhateveringForScience@Home*. The host organization can create their own massive supply of *WhateverCoins* and send them out on a regular basis to users who are doing computations for them, in proportion to their computational effort. These coins can then be traded for MC_2, etc. based upon their scarcity.

Appendix I: Lightweight Client and Ledger System

Clients will have the option to select usage of the lightweight client if they so prefer, which will use much less disk space than the full-weight client which contains the entire blockchain. Please note that the lightweight client disables use of colored coins. Lightweight clients will use the following ledger file to keep track of transactions,

1. Block header hashes used as PoW for every block and their block heights. These are stored sequentially.
2. All addresses containing more than 0.0001 coins, the quantity of spendable coins in these addresses (minus coins deemed unspendable by stake submission transactions or conditional transactions), the quantity of unspendable coins in the address, the block in which the coins become unspendable and duration of unspendability (stake submission transactions and conditional transactions only), and the block heights in which the number of coins in the address last changed. The addresses are sorted by the value of the address. These are stored as a B+ tree to faciliate item retrieval, insertion, and deletion operations.
3. All currently submitted and unredeemed stakeholder transactions and their tickets (if applicable). These are stored as B+ trees, by the value of the submitting address.

Once a new block has entered the network and has been verified by the PoS miners, the lightweight client will insert the new transaction data afforded by the block into the ledger and hash it (SHA3-256) to verify its integrity (confirm that the ledger hash in the block header matches the ledger hash found by the client).

Lightweight clients will still operate as relays on the network for blocks and transactions, but only for transactions it can see as valid (e.g., if an address contains 0.000010 coins and these coins are sent to another address, a lightweight client will not relay the transaction because it is unable to see whether or not this address actually possesses this quantity of coins).

To enhance security, the last 256 blocks will be kept in the client's memory for reference. Reorganizations of up to these lengths are then still able to occur, by rolling back the transactions recorded in the orphaned chain.

Appendix J: Algorithms for Proof-of-Stake Voters to Assess the Validity of Proof-of-Work Blocks

PoS voters can decide themselves how best to assess the validity of blocks submitted to the network, but the client itself will ship with simple metrics to allow for automated voting on PoW blocks based on,

1. Sigmoidal distribution test of relayed transactions.
2. Correct ordering of the transaction tree based on type, fee, and address.
3. Inclusion of the stake votes from the previous block (SHA3-256 hash of the stake sub-blocks corresponds to the stake votes they saw cast on the network).
4. Fees are included where applicable.
5. All difficulties are met (PoW, PoS submission).
6. No double-spend transactions are observed.

Sigmoidal Distribution Test
This test examines incoming transactions based on the time in which they enter the network, and assumes that it is extremely unlikely transactions that are propagated from a random node within the network will not be seen by all other nodes in the network within approximately 30 seconds (in the BTC network, the average time of propagation is just 3.354 seconds) [2].

Assume we have a sigmoid function defined as,

$$\zeta x = \frac{1}{1 + e^{-18k+s}}, \tag{14}$$

where the x-axis refers to time in minutes from now and when we first saw the transaction on the network, and the y-axis refers to the transaction weight value (ζ). So, any value of x is then (current local time) - (local time transaction was seen on network). k is the steepness factor, by default 18 in the client. s is the shift of the function along the y-axis, by default 5.8 in the client.

EXAMPLE: Our client with a PoS ticket observes that exactly 0.17 minutes ago a transaction entered the network. This transaction is given the weight of 0.061 from the sigmoid function above.

Our local client sees a new PoW block submitted to the network containing many transactions. Some of these transactions will be observed by the client, while others will not be.

1. All transactions seen by both the client and the person submitting the PoW block are given a transaction score of 0 and are effectively ignored.
2. All transactions seen by the PoS mining client but not seen in the PoW block or that contradict a transaction seen in the PoW block are given a transaction score of ζx.
3. All transactions that exist in the PoW block that do not exist in the PoS block are ignored.

NOTE: Because of 3., it is possible that PoW miners can hide transactions from the network. Zero-confirmation transactions are still not entirely safe from double spend. For instance, if a zero conf Tx has gone through the network and 60s has passed, the PoW miner can attempt to incorporate a different transaction in their block and invalidate the publically broadcast transaction. However, the PoS miners will have already seen the public transaction and will reject the block submitted by the PoW miner. As with a normal BTC zero conf doublespend, the transaction will be invalidated.

NOTE: Dealing with double-spends. When a PoW miner and a PoS miner see one or more double or multiple-spend transactions on the network, all of these transactions should be discarded by the miners and the address from which the spend is being performed blacklisted until the next block. Clients in general should still propagate both transactions, so that the entire network can quickly see that they both should be invalid. This, to some extent prevents the problem as seen in the above note of a malicious entity sending two different transactions to the PoW and PoS miner that contradict, forcing the

PoS miner to reject the PoW block's transactions. The sigmoid function also prevents a network attack in which a malicious entity sends several spends from the same originating address close to when the next block is solved in the hopes that the PoS miner will see some transactions the PoW will not see within a span of time large enough to affect the vote. This will likely never happen with the sigmoid function above, which is calibrated to weight heavily differences of 30 seconds or more and lightly those of less than 30 seconds.

Now, we sum the transaction weight values and divide by the number of transactions to afford a *TRANSACTION LIKELIHOOD SCORE*,

$$\sum TRANSACTION\ SCORE(x)\ for\ all\ transactions, \tag{15}$$

where $TRANSACTION\ SCORE(x) = TRANSACTIONSCORE$ as defined by the rules above and the ζ function $x = TRANSACTIONTIMEDIFFERENCE$.

The client will then be able to set a cutoff score for the *TRANSACTION LIKELIHOOD SCORE*. By default, the transaction likelihood cutoff is 0.75. When a new block comes in, all transaction likelihood scores of less than the cutoff will cause the client to automatically vote in the block, while any score of 0.75 or greater will cause the client to automatically reject the incoming block by vote.

NOTE: Cutoff score and $\zeta(x)$ function steepness factor k as well as shift s need to be modulatable in the client software. As described here, the functions with their default parameters are not intended to scale with the size of the network to extremes. Depending on the average network transaction propagation times, these parameters may require tweaking. Experimental data should establish what these values should ideally be for different transaction volumes and network sizes.

EXAMPLE: A table of *TRANSACTION TIMESS*, *TRANSACTION SCORES*, and the *TRANSACTION LIKELIHOOD SCORE*(TLS),

Tx Time/x	ζx	**PoW Contains**	**Tx Score**
0.01	0.0036	N	0.0036
0.02	0.0043	N	0.0043
0.17	0.061	N	0.061
0.3	0.4	N	0.4
0.56	0.99	Y	0
0.9	1	Y	0
1	1	Y	0
1.1	1	Y	0
1.5	1	Y	0
1.7	1	Y	0
		TLS	0.4689

Tab. 2. Transaction Times, Transaction Scores, and Transaction Likelihood Score

In this example, this set of transactions will cause the client to vote 'Yea' on a block because the TLS is less than the default 0.75 cutoff.