HARDENING THE HIVE: OPTIMISING CHAINWORK SCALING IN LITECOIN CASH

Version 1.0

Iain CRAIG, Sebastian CLARKE, Aleksandrs LARIONOVS, Michał WYSZYŃSKI, Thomas BEASLEY, Kasper Brandt HANSEN, Joseph Paul MCMAHON, Maciej ZAMS, Federico DE GONZÁLEZ-SOLER, Ilya HELLE, Rabin LAMICHHANE

Litecoin Cash Developers, London

Abstract. An improved calculation for block chainwork on a hybrid proof-of-work/hivemining blockchain is presented. This calculation improves cooperation between miners using proof-of-work and hive miners, while further mitigating pure proof-of-work attacks, as well as two classes of hybrid proof-of-work/hive attack. The chainwork scaling mechanism is likely to be of value to other types of hybrid blockchain.

1. Background and motivation

In the months following activation of the original Hive Mining[1] consensus rules on the LitecoinCash mainnet in Feb 2019, various issues were observed:

1.1 Block type distribution ratio

The ideal block type distribution ratio of 50:50 between proof-of-work and hivemined blocks was not being reached. In practice, typically 33% of blocks in an arbitrary timespan were seen to be hivemined.

1.2 Shallow depth orphans

Although shallow depth orphan chains of 1-2 blocks were entirely expected (especially since, after a proof-of-work block, all hivemining beekeepers try to find a block simultaneously), users were dissatisfied with the particular situation of seeing Hive blocks get orphaned at depth 0 by proof-of-work blocks.

1.3 Deep reorganisations in July 2019

In July 2019, deep reorgs of up to 102 removed blocks were observed, indicating a successful double spend via a classical withheld-blocks attack[2][3]. Analysis of both the orphaned chain and the chain which caused the reorganisation showed that the attacker had attempted either a pure proof-of-work or a “stealthy bees” attack.

In the first divergent block, the attacker created bees. Only in the following block did the attacker's double spend occur. This suggests that the attacker was hoping to stay ahead of the mainnet in chainwork terms long enough to mature those bees on the private chain.

These attacks occurred when the hivemining Global Index on mainnet was around 100, indicating full effectiveness of hivemining protection.

Rather than getting ahead on chainwork immediately, there was a significant lag period as the attacker ramped up their proof-of-work difficulty. The additional chainwork provided to mainnet by hivemining prevented the attacker overtaking mainnet in chainwork for 40 blocks.

The attacker's final difficulty of 4X that of mainnet suggests the attacker could bring to bear at least 400% of the hashpower of the entire of mainnet, making this at least an "80% attack".

While the original Hive rules are capable of protecting the network indefinitely against an attack with equivalent power to the entire of mainnet, they were not sufficient to defend against an attack of the magnitude seen.

If the attacker had intended to mature the bees they created on their private chain in order to gain bonus chainwork from hivemining (a "stealthy bees" attack), this did not come to pass as the withheld blocks were broadcast to the network long before the bees would have been able to mature.

2. Dynamic chainwork scaling

In order to provide hivemined blocks with sufficient weight to assist in securing the blockchain, when their "natural" chainwork would be very low due to hivemining solving difficulty being many orders of magnitude lower than typical proof-of-work difficulty, the original Hive consensus rules award bonus chainwork to each hivemined block based on the chainwork of the proof-of-work block behind it.

$$chainWork = \sum_{n \in pow\ blocks} expectedHashesToCalculate(blocks[n].difficulty)$$

$$+ \sum_{n \in hive\ blocks} k * expectedHashesToCalculate(blocks[n-1].difficulty) + (MAXINT - blocks[n].beeHash)$$

In the original rules, the scaling parameter *k* was set to 1, allowing a hivemined block to inherit the same amount of chainwork as the preceding proof-of-work block (plus the very small amount of natural chainwork the hivemined block itself contributes).

It was observed that this scaling factor value was not high enough to indefinitely protect the network against a pure proof-of-work attacker with an amount of hashpower exceeding that of the rest of the network.

Simulations showed that with high enough value for *k*, hivemined blocks could protect against very high amounts of hashpower – many times greater than the hashpower of the rest of the network.

However, awarding massive chainwork bonuses to hivemined blocks potentially weaponizes hivemining by giving a single hivemined block enough chainwork to reorg many proof-of-work blocks. This dangerous imbalance could create a slew of new attack vectors.

Therefore, it was decided to introduce a dynamic value for *k.* By awarding bonus chainwork to both hivemined blocks and proof-of-work blocks, *so long as they are alternating*, every block on the public network can gain much more chainwork than every block on an attacker's withheld chain, whether the attacker is using a pure proof-of-work attack, a "stealthy bees" attack, or a "lazy bees" attack (discussed below).

This enables both proof-of-work miners and beekeepers to work symbiotically, giving each more power to secure the network while not enabling either to overwrite the work of the other via disproportionate chainwork.

Giving a slightly higher bonus to hivemined blocks than to proof-of-work blocks helps to prevent the shallow hive block orphans described in section 1.2.

For a hivemined block, the base value for k is 2 (`consensus.minK`). This value can scale up to k=16 (`consensus.maxK`), depending on current hive difficulty. Maximum value is reached at a hive difficulty of 0.006 (`consensus.maxHiveDiff`). This hive difficulty is typical of that seen on mainnet with GI at 100 and no adverse network conditions.

For a proof-of-work block, the highest value for k is 5 (`consensus.maxKPow`) -- but this value is only awarded to a proof-of-work block that immediately follows a hivemined block. Each subsequent proof-of-work blocks with no hivemined block diminishes this value by 1, so after 4 proof-of-work blocks in a row, no further chainwork bonuses are gained by proof-of-work blocks until a hivemined block follows.

A proof-of-work block's k is also scaled in a quadratic relationship with current hive difficulty before being applied, so that in low hive difficulty conditions no chainwork bonus is awarded to proof-of-work blocks.

Block type	Difficulty (Hive or pow, as applicable)	*k*
Hive	0.004070537	11
Pow	69277979.77	4
Pow	78589530.27	3
Hive	0.004070537	11
Pow	76446796.11	4
Pow	76596414.25	3
Pow	78150481.51	2
Hive	0.00383113	10
Pow	77645028.36	4
Pow	78461351.11	3
Pow	71891500.66	2
Hive	0.003605782	10
Pow	70335474.37	4
Hive	0.003846211	10
Pow	68338147.06	4
Hive	0.004102633	11
Pow	69670305.51	4
Hive	0.004376151	12
Pow	71918631.69	4
Hive	0.004667969	12
Pow	75944216.26	4
Pow	75702073.85	3
Pow	88516773.47	2

An example of dynamic k-scaling, simulated with live data from mainnet

A simple simulator was devised to allow us to replay the attacker's attack run with these new rules; this can be found at https://www.litecoinca.sh/k-optimiser/. With the final Hive 1.1 parameters of minK = 2, maxK = 16, maxHiveDiff = 0.006, maxKPow = 5, powSplit1 = 0.005, powSplit2 = 0.0025, it can be seen that the attacker is never able to get ahead during this run.

In fact, from the beginning of the attack the attackers falls behind immediately and the gap only continues to widen.

Note that this simple simulator does not consider relative time differences in block issue between the withheld chain and mainnet chain.

We consider this symbiosis between mining methods to be beneficial to the security of any hybrid blockchain that features different types of block production.

3. Improved hive difficulty adjustment

In order to address the fact that the ideal hive block to proof-of-work block ratio of 50% was not being achieved on mainnet, we present a revised approach to calculating hive difficulty, in combination with a relaxation of the rule that hive blocks must follow proof-of-work blocks.

Due to the constraint that a hive block necessarily follow a proof-of-work block, the maximum achievable hive to proof-of-work ratio was 50%. This presents a problem for difficulty adjustment, as it leaves no room for a “hive is too easy” signal, beyond hitting the desired 50% ratio. This means that difficulty adjustment in this context by necessity must treat the ideal situation as the “hive is too easy” signal and starting increasing the difficulty, until such time as we do not see the ideal 50% ratio, before slowly decreasing it to target the optimum ratio again. The obvious result is that whenever the difficulty adjustment is tuning the hive difficulty to the online population of bees, the number of hive blocks seen must be less than the ideal ratio.

In practice, in order to keep up with natural variations in the online bee population, this facet of the difficulty adjustment has resulted in a hive to proof-of-work ratio of closer to 33%.

A further reason why difficulty adjustment in a hive mining context is different to that in proof-of-work is due to the nature of the “signal” that the difficulty adjustment algorithm has to work with.

In proof-of-work mining, the speed at which blocks are solved gives a scalar representation of the imbalance between the network hash rate, and the ideal block times: Individual block times, when averaged over many samples due to the probabilistic nature of proof-of-work solving, give a linear indication of how much the network is over or under-performing, with respect to target block times.

In contrast, with Hive mining there is no temporal component: If a hive is able to solve a block, it can do so immediately. A single solve, therefore imparts no relevant data about the network conditions and how they relate to the desired hive difficulty, ergo, designing a difficulty algorithm for hive mining around block times is clearly meaningless and another source for an indicative signal is needed.

As the presence of a solved hive block alone is quite a noisy signal when compared to an individual proof-of-work block solve time, a reasonable number of blocks must be examined to achieve a stochastically valid sampling. In order to allow such a sampling to indicate whether there are too few or too many hive blocks appearing with respect to the ideal ratio and current difficulty, we have chosen to relax the constraint that a hive block must follow a proof-of-work block.

By allowing the network to wander into the territory of “too many hive blocks”, the difficulty adjustment can work from both sides, with the result being that over time, the average hive to proof-of-work ratio will be much closer to the configured target of 50%.

Given the above, the difficulty adjustment algorithm itself is rather uncomplicated, an implementation of a simple moving average based on Zawy’s Universal Difficulty Algorithm[1] with $w = 1$, $r = 1$. The key difference being that the moving average samples a constant number of hive blocks (therefore, a variable window size on the actual blockchain, depending on the observed hive to proof-of-work ratio), and calculates its difficulty adjustment based on the difference between the

observed hive/proof-of-work ratio in the window, and the target, rather than the average solve times and target solve time as in proof-of-work.

On mainnet, the amount of hive blocks that make up a difficulty sampling window is set at 36, and 2 hive blocks are allowed to follow a single proof-of-work block. Note that, due to the linear adjustment present in the simple moving average, and the maximum hive/proof-of-work ratio in the sampling window being 2:1 (two hive blocks for every proof-of-work block), the difficulty algorithm can correct less aggressively when there are too many hive blocks, than when there are too few.

A modified implementation was considered in order to correct this behaviour. However, the simple version has been used for two reasons. The first is clarity of implementation, the second is that a less aggressive adjustment when difficulty is too easy, will at worst result in slightly increased shallow hive orphans, and more frequently, in slightly more hive blocks being produced than the target, which is more preferable to the users of the network than the opposite.

4. Effect on attackers

4.1 Pure proof-of-work attackers

As described in Section 2, a pure proof-of-work attacker (ie, one with no bees) is immediately disadvantaged from the moment they withhold their first block. Whatever the public network hive difficulty at the start of their run, their proof-of-work chainwork bonus will disappear within 4 private blocks.

As the attacker ramps their proof-of-work difficulty in an attempt to get ahead in chainwork, a level of hashpower of up to 10 times the entire rest of the network will ceiling out difficulty before they can do so.

4.2 Attackers with privately matured bees (“stealthy bees” attack)

This attack envisages an attacker who initially has no bees and starts with a pure proof-of-work attack. In their first withheld block, the attacker creates bees with the intention of maturing them on their withheld chain and then using them to accumulate chainwork.

However, the bee maturation period of 1152 blocks and the rapid divergence of chainwork between the public and withheld chains makes it very unlikely that the attacker’s withheld chain would be viable after this maturation period.

4.3 Attackers with publically matured bees (“lazy bees” attack)

This attack refers to a hypothetical attacker who has bees they have matured on the public network in advance of their attack run. The attacker intends to produce a viable withheld chain through a combination of proof-of-work mining and hivemining.

Supposing that the attacker does not own a large majority of the total bee population on the public network (ie a strong majority hiveweight), then when they initially produce their first private block, it is unlikely that their hive will be able to meet the hive difficulty target (inherited from the last shared block on the public chain).

By the time hive difficulty on their private chain falls enough for their private bees to be able to start to produce blocks, the interim proof-of-work blocks they have been forced to produce will have quickly lost chainwork bonus and be falling behind mainnet.

In addition, once these “lazy bees” do begin producing blocks, hive difficulty will necessarily have fallen so low that neither those hivemined blocks, nor proof-of-work blocks interspersed with them, will gain significant chainwork bonuses.

5. Acknowledgements

With thanks to Antoine Brule, Auscoin, uross, DOTTAT and all others who helped with design, implementation and testing.

6. References

[1] *The Hive: Agent-Based Mining in Litecoin Cash.* The Litecoin Cash Foundation (2018). https://litecoinca.sh/downloads/lcc_whitepaper.pdf

[2] *Double-spending fast payments in bitcoin*. Karame, G.O., Androulaki and E., Capkun, S. (2012). In CCS '12 Proceedings of the 2012 ACM conference on Computer and communications security, pp 906-917.

[3] *Litecoin Cash (LCC) was 51% attacked*. James Lovejoy (2019). https://gist.github.com/metalicjames/82a49f8afa87334f929881e55ad4ffd7

[4] *"Universal" Difficulty Algorithm.* Zawy12 (2019). https://github.com/zawy12/difficulty-algorithms/issues/41