GhostRider Mining Algorithm

By
Tri Nguyen-Pham

I. **Objective:** Create an alternative mining algorithm that is highly resistant to asics as well as minimize the effect of fpgas and heighten the entry point cost for fpga mining significantly.

Technology:

GhostRider is a combination of known mining technologies and methodologies from x16r (Raven) and CryptoNight (Monero). X16r provides a randomness to an existing hash chaining methodology for mining, it lacks a memory requirement which means asics can potentially gain significant advantages over gpus. CryptoNight, on other hand has features that require cpu/gpu memory which makes it harder for asics to gain a significant advantage over cpu/gpu, but it lacks the randomness that x16r has.

Over the recent year, the Monero team committed to combat asics by forking CryptoNight to add more variables to its memory requirements, as well as hashing methodology. However, each fork's hashing method remains static.

GhostRider methodology:

With the realization of the value that the x16r randomness provides in battling the curve of asic efficiency combined with the impact of a high memory requirement. The concept of GhostRider was born by combining both methodologies together by randomly selecting 15 different core base algorithms and mixing them with 3 different random variants of Cryptonight hashing. These algorithms are divided into 3 groups of 5 random order core algorithms followed by 1 random order CN variant. All 15 order core algorithms are random but not no single algorithm being repeated in the same chain. The same goes for the order of CN derivaties.

Random ordering algorithm: To archive pre-deterministic ordering, the algorithm uses previous block hash nibbles in order from right to left to determine what algorithm to hash next for the 15 core algos. Each nibble is a single hex digit(0-F) and there are 64 nibbles in a block hash. If a nibble hex is F(15 in decimal) then it wraps around as 0. See hex number to algo map below. If a hex digit has seen before in the previous nibbles, it moves to next nibble in the hash. The process is repeated until all 15 unique hexes are selected. Similarly, CN variant ordering is determined by hex digit and _modified_.

Hex to algo maping:

0 or F-Blake
1-Bmw
2-Groestl

3-Jh
4-Keccak
5-Skein
6-Luffa
7-Cubehash
8-Shavite
9-Simd
A-Echo
B-Jamsi
C-Fugue
D-Shabal
E-Whirlpool
F-Sha512

Example:

Given previous block hash is;
0000135e13882a45caa301fc03429e416e7ce8d8edebdffe495ab337f9c98582
Going from right to left we have: 2-Groeslt, 8-Shavite, 5-Skein, 8(skip), 9-Simd, c-Fugue, 9(skip), f-Blake, 7-Cubehash, 3-Jh, 3(skip), b-jamsi, a-Echo, 5(skip), 9(skip), 4-Keccak, e-whirlpool, f(skip), f(skip), d-Shabal, b(skip), e(skip), d(skip),e(skip), 8(skip), d(skip), 8(skip), e(skip), c(skip), 7(skip), e(skip), 6-luffa, 1-Bmw. Stoo, 15 algo and order hash has been selected as follows: **Groeslt->Shavite->Shein->Simd->Fugue->Blake->Cubehash->Jh->jamsi->Echo->Keccak->whirlpool->Shabal->Luffa->Bmw.**
Now similarly for CN variants, we go from right to left of previous block hash but this time we hex mod 3 + 2 so this is what we get.
2-CNv4, 8(skip), 5(skip), 8(skip), 9-CNv2,c-(skip), 9(skip), f(skip), 7-CNv3. Stop and now we have the CN variants ordering as follow. **CNv4->CNv2->CNv3**.
Put algo ordering and CN ordering in 3 groups which each group contains 5 algorithms and 1 CN variant we get
Groeslt->Shavite->Shein->Simd->Fugue->CNv4->Blake->Cubehash->Jh->jamsi->Echo->CNv2->Keccak->whirlpool->Shabal->Luffa->Bmw->CNv3