Algoritmo di Mining GhostRider

By
Tri Nguyen-Pham

I. **Obbiettivo:** Creare un algoritmo di mining alternativo che sia altamente resistente agli asic e minimizzi contemporaneamente anche gli effetti delle fpga e ne alzi significativamente il costo di entrata.

Tecnologia:

GhostRider è una combinazione di tecnologie di mining già conosciute e metodologie prese da x16r (Raven) e CryptoNight (Monero). X16r provvede a una casualità nell'esistente metodologia di hash chaining per il mining, ma gli manca un requisito di memoria il che significa che gli asic possono prenderne un significativo vantaggio sulle gpu. CryptoNight , d'altra parte ha delle caratteristiche che hanno bisogno di memoria Cpu/Gpu ma gli manca la casualità che ha x16r. Nei recenti anni , il Team di Monero si è impegnata a combattere gli asic forcando CryptoNight aggiungendo più variabili ai suoi requisiti di memoria e anche metodologie di hashing. Comunque ogni fork che ha fatto è rimasto statico.

Metodo GhostRider:

Il valore che la casualità x16r fornisce abbattendo la curva dell'efficienza degli asic combinato con l'impatto di un requisito di memoria molto alto.

Il concetto di GhostRider è nato combinando entrambe le metodologie insieme selezionando casualmente 15 algoritmi di base e mescolandoli assieme casualmente a 3 diversi varianti di algoritmi CryptoNight.

Questi algoritmi sono divisi in 3 gruppi di 5 algoritmi casuali in ordine seguiti da 1 algoritmo CN casuale. Tutti i 15 algoritmi sono casuali ma nessuno algoritmo viene ripetuto nella stessa catena. Lo stesso vale per gli algoritmi CN.

Ordine casuale degli algoritmi: Per archiviare l'ordinamento pre-determinato, l'algoritmo usa un nibble (quattro bit) del blocco precedente in ordine dalla destra alla sinistra per determinare il prossimo algoritmo da usare per i 15 del core. Ogni nibble è una singola cifra esadecimale (0-F) e ci sono 64 nibble in un hash di un blocco. Se cè una F nella cifra (15 in decimale) allora viene contata come uno 0. Nella tabella sotto si possono vedere gli accoppiamenti numero esadecimale–algo. Se una cifra esadecimale c'era già nella nibble precedente, allora si va avanti alla prossima nibble nella hash. Il processo è ripetuto finchè tutte i 15 esadecimali sono stati assegnati. Similmente, l'ordine delle varianti CN è determinato da numeri esadecimali e modificato.

Tabella Esadecimale-Algo:

0 o 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

Esempio:

L'hash del blocco precedente è:
0000135e13882a45caa301fc03429e416e7ce8d8edebdffe495ab337f9c98582
Andando da destra a sinistra abbiamo: 2-Groeslt, 8-Shavite, 5-Skein, 8(salta), 9-Simd, c-
Fugue, 9(salta), f-Blake, 7-Cubehash, 3-Jh, 3(salta), b-jamsi, a-Echo, 5(salta), 9(salta), 4-Keccak, e-whirlpool, f(salta), f(salta), d-Shabal, b(salta), e(salta), d(salta),e(salta), 8(salta), d(salta), 8(salta), e(salta), c(salta), 7(salta), e(salta), 6-luffa, 1Bmw. I 15 algo e il loro ordine è stato scelto così: **Groeslt>Shavite->Shein->Simd->Fugue->Blake->Cubehash->Jh->jamsi->Echo>Keccak->whirlpool->Shabal->Luffa->Bmw.**
Ora similmente per le varianti CN andiamo da destra a sinistra nel blocco precedente ma stavolta facciamo un calcolo mod 3+2 all'esadecimale e questo è quello che abbiamo:
2-CNv4, 8(salta), 5(salta), 8(salta), 9-CNv2,c-(salta), 9(salta), f(salta), 7-CNv3.
Si ferma e abbiamo ordinato le varianti CN in questo modo: **CNv4->CNv2->CNv3**. Mettiamo gli algo in ordine con CN in 3 gruppo dei quali ogni gruppo contiene 5 algo e 1 variante CN e il risultato
Groeslt->Shavite->Shein->Simd->Fugue->CNv4->Blake->Cubehash->Jh->jamsi->Echo->CNv2->Keccak->whirlpool->Shabal->Luffa->Bmw->CNv3