GhostRider Louhimis Algorytmi

kirjoittaja:
Tri Nguyen-Pham

I. **Tavoite: Luoda alternatiivinen louhimisalgorytmi, joka on todella vastustuskykyinen** ASIC:lle, sekä minimoida FPGA:iden efektit nostaen toimintakulujan huomattavasti. mining significantly.

Teknologia:

GhostRider on yhdistelmä tunnettuja louhintateknologioita ja metodeita x16r:stä (Raven) CryptoNightiin (Monero). X16r mahdollistaa sattumanvarausuuden olemassaolevaan tarkistelinkitysmetodologiaan louhimista varten, muistivaatimuksen puuttuessa, joka tarkoittaisi sitä, että asicit voisivat saada huomattavan etulyöntiaseman näytönohjaimiin nähden. CryptoNightissa taas on sellaisia ominaisuuksia, jotka vaativat prosessorin/ näytönohjaimen muistia, joka tuottaa asiceille suuria vaikeuksia saada etulyöntiasemaa prosessori-/näytönohjainlouhijiin nähden. Siltä taas puuttuu x16r:än sattumanvaraisuus. Viime vuoden aikana Moneron tiimi on omistautunut asiceja vastaan taistelemiseen haara-uttamalla CryptoNightin lisäten muuttujia sen muistivaatimuksiin, sekä muuttamalla tarkistussumman metodologiaa. Toisaalta, jokaisen haaran tarkistemetodi pysyy staattisena.

GhostRider metodologia:

Tajusimme x16r:än satunnaisuuden arvon tässä taistelussa asicejen kehityskurvia vastaan yhdistettynä korkeaan muistivaatimukseen. Ghostriderin konsepti syntyi Yhdistämällä molemmat metodologiat sattumanvaraisesti valitsemalla 15 eri ydinpohjaista algorytmiä ja sekoittamalla ne kolmen eri sattumanvaraisen Cryptonightvariantin kanssa yhteen. Nämä algorytmit ovat jaoteltuna kolmeen eri ryhmään, joissa kaikissa on 5 sattumanvaraista ydinalgorytmiä, joita seuraa yksi sattumanvarainen CN variantti. Kaikkien 15 ydinalgorytmejen järkestys on sattumanvarainen, mutta sama ei koskaan toistu kahdesti samassa ketjussa. Sama pätee CN derivatiivejen järjestykseen.

Sattumanvarainen algorytmin järjestyksen määritys: ennaltamääritty algorytmi käyttää viime lohkon varmisteen "nibblejä" järjestyksessä vasemmalta oikealle määrittääkseen mitä algorytmejä määrätä seuraavaan 15 ydinalgorytmiin. Jokainen "nibble" on yksittäinen hex-luku(0-F) ja yhdessä lohkossa on 64 nibbleä.Jos nibblen hex on F(15 desimaaleissa), sitten se kääntyy uudestaan nollaan. Ks alapuolella oleva hexnumerokartta. Mikäli hex-luku on nähty aiemmin nibblessä, se siirtyy varmisteen seuraavaan nibbleen. Prosessi toistuu, kunnes kaikki 15 uniikkia hexiä on valittu. Samalla tavalla määritetään myös CN varianttejen järjestys hex-luvun mukaan ja tulee muokatuksi.

Hexistä algorytmiin kartoitus:

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

Esimerkki:

Viime lohkon varmiste on;
0000135e13882a45caa301fc03429e416e7ce8d8edebdffe495ab337f9c98582
Vasemmalta oikealle mennessä meillä on: 2-Groeslt, 8-Shavite, 5-Skein, 8(ohi), 9-Simd, c-Fugue, 9(ohi), f-Blake, 7-Cubehash, 3-Jh, 3(ohi), b-jamsi, a-Echo, 5(ohi), 9(ohi), 4-Keccak, e-whirlpool, f(ohi), f(ohi), d-Shabal, b(ohi), e(ohi), d(ohi),e(ohi), 8(ohi), d(ohi), 8(ohi), e(ohi), c(ohi), 7(ohi), e(ohi), 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.**
Seuraavaksi samalla lailla CN varianteille, menemme vasemmalta oikealle viimeisen lohkon varmisteessa. Tällä kertaa hex modaamme 3 + 2, joten saamme tällaiset vastaukset.
2-CNv4, 8(ohi), 5(ohi), 8(ohi), 9-CNv2,c-(ohi), 9(ohi), f(ohi), 7-CNv3. Loppu.
Nyt meillä on seuraavat CN variantit seuraavassa järjestyksessä. CNv4->CNv2->CNv3. Laita algorytmijärjestelijät kolmeen eri ryhmään, joissa kaikki sisältävät 5 algorytmiä ja 1 CN variantin. Saamme siitä:
Groeslt->Shavite->Shein->Simd->Fugue->CNv4->Blake->Cubehash->Jh->jamsi->Echo->CNv2->Keccak->whirlpool->Shabal->Luffa->Bmw->CNv3