Future-proofing egalitarian mining - Unprll

Christopher Tobias

November 28, 2018

1 Introduction

Cryptonote[1] has been a leader in the cryptocurrency field, enabling confidential transactions, egalitarian mining methods, scalability and other features in a completely new codebase. Its success is found in the existence of Cryptonote-based coins.

However, with the advancement of technology, Cryptonote's Proof-of-Work mechanism (PoW) is slowly losing its egalitarian property, leading to possible centralization of mining power.

In this paper, we study Cryptonote's PoW and propose a new Proof-of-Work algorithm that will lead to fair mining for all. The result of this paper is a new electronic cash called "Unprll" (short for UNPaRaLLeled. Ticker: ULL).

2 An analysis of Cryptonote's PoW

2.1 Strength

Cryptonote uses a memory-bound hash function called Cryptonight[2]. It is a combination of other hash functions like Keccak, Blake, Skein along with the use of AES. In coins like the reference CryptoNote-Coin et al, a memory requirement of 2MB is mandated. This prevents FPGA and ASIC mining because the memory requirement is prohibitively large, while allowing CPUs to fit the buffer in L3 CPU cache. Further, heavy random access to memory is required for the hash function, thereby putting the sequentially-accessible GPU memory at a disadvantage.

2.2 Weakness

However, memory-boundedness cannot guarantee egalitarian mining. As an example, High Bandwidth Memory (HBM)[3] was introduced to GPUs. As a result, hash rates of upto 2KH/s and more could be attainable with a single GPU, at profitable energy consumption. Making a rig of these leads to massive hash rates attained by a single miner.

Another line of attack comes from marketplaces for selling and buying compute power for mining. These introduce massive hash power into the network, creating a difficulty surge from which other miners may not recover quick enough.

As of the time of writing, Cryptonight ASICs entered the mining scene, forcing coins to come up with alternatives.

3 The Unprll PoW

Unprll tries to make CPU mining the only efficient method. This section outlines the idea behind the mining method implemented in Unprll.

3.1 Review of compute units

The different compute units used in cryptocurrency mining have different characteristics:

- CPU: General purpose compute unit. Consists of few fast units. Meant for general purpose tasks. Best at tasks involving random memory access and performing tasks one after another.

- GPU: Graphics oriented compute unit. Consists of multiple slower units. Can run parallelized tasks (such as drawing and physics simulations) efficiently.
- FPGA: Generic, rewirable units. Best oriented towards running tasks that don't depend on memory.
- ASIC: Special purpose, highly efficient units. Best oriented towards running tasks that don't depend on memory.

In order to make CPUs the only efficient miners, it is necessary to exploit features characteristic only to CPUs. Two points pop out from the above review:

1. CPUs are the most effective at using memory in a random order. This is used by Cryptonote in the Cryptonight hash function.
2. CPUs are most effective at running tasks one after another (i.e. in sequential order).

The second characteristic is used in the PoW implemented in Unprll (the namesake of the coin). In short, the procedure for mining Unprll is such that it is pointless to run multiple miner instances (via multithread, multiprocess, multisystem et al). This ensures that only one miner is effectively mining for any one wallet.

3.2 Method

The technique used in Unprll is by starting with the hash of the block, and continually running the hash function on the previously generated hash till the generated hash satisfies a given difficulty. Since each hash generated depends on the previous hash, there is no scope of multithreading a single miner. The hash function used is Cryptonight (with certain changes as mentioned below), to ensure FPGA and ASIC resistance.

To prevent multiprocess/multisystem mining, the following measures were implemented:

- Changes to the block template:
  - The nonce field is removed.
  - Fields for the number of iterations, the hash_checkpoints, and the miner_specific value were added.
- The starting template for miners was changed to consist of the following:
  - The major and minor version of the block
  - The previous block ID
  - The miner_specific value
  - The approximate timestamp (accurate to a block interval)
  - The Merkle root of the transactions in the block excluding that of the miner transaction.
  - The number of transactions in the block.

The miner_specific value mentioned above is a value specific to any one miner. In the case of Unprll, the public spend key of the miner is used. For every Unprll wallet, there are two addresses:

- The standard address as generated in Monero
- A truncated address as mentioned in the Cryptonote whitepaper, which consists of just the public spend key. The secret and public view keys are generated from the public spend key.

The mining reward is sent to the truncated address and is recognized under the standard address. While this leads to on-chain linkability of blocks to miners, as long as the primary address of the wallet (known as the Mining Address in Unprll) is not exposed, it does not pose a privacy breach. This is enforced with the use of subaddresses.

This results in every miner getting the same starting template, only differing in the miner_specific value. This provides sufficient randomization to ensure that there isn't a race condition where the fastest miner always wins.

3.3 Verification

Verification presents an interesting problem. Since the mining process involves hashes that potentially have no correlation to the block, there is a risk of forgery of the proof of work. Hence, the verifier repeats the mining process, generating hashes and (at checkpoint intervals) comparing the hash generated with the hash listed in hash_checkpoints. Verification can be multithreaded in contrast to mining, because different threads can check different sections of hash_checkpoints.

Verification proceeds as follows:

1. Check if the last hash is valid
2. Check if the miner_specific value is valid. In the case of Unprll, this is performed by generating the truncated address and checking whether the miner transaction is sent to that address.
3. Check if the number of checkpoints doesn't exceed the number of iterations
4. Keep generating hashes until one of the following happens:
    - A hash earlier than the last hash is valid for a given difficulty (invalid).
    - A checkpoint hash does not match a hash generated from the previously generated hash (invalid).
    - The number of verification iterations matches the mining iterations (valid).

3.3.1 Improvement

An improvement in verification speed is possible, however it involves results that are probabilistic rather than deterministic. The reasoning is if a large number of checkpoints were verified, it is highly likely that enough work was done. This is valid for the following reasons:

- Inversion of a hash function is computationally intensive. Adding to the difficulty is the need for a miner to list the hashes leading to the valid proof of work. Hence, an attacker cannot forge a valid proof of work without wasting an immense amount of computing power in generating a chain of hashes in reverse.
- If the attacker tries to generate partial chains of hashes in the proof of work, as the number of nodes in the network increase, he is at risk of getting his forgery detected and invalidated.

To spread the load of verification, each node verifies a small section of the hash_checkpoints at random. If any one node discovers a forgery, a message is broadcast to the network, specifying which checkpoint failed verification, leading to invalidation of the block.

4 Cryptonight changes

To make Cryptonight resilient against FPGAs and ASICs, the following changes were made to create the RNJC (RNJC's Not Just Cryptonight) hash function:

- Recursion was introduced into the hash function. There are 1024 rounds of the mixing loop followed by 4 rounds of calling itself, to a depth of 2. This increases the amount of memory required, weakens GPU implementations and adds complexity to hardware implementations.
- The encryption algorithm used was replaced from the AES Standard Rijndael to AES finalist CAST-256 based on a comparison of the 15 AES candidates[4] for the following reasons:
    - CAST-256 follows a different structure to that of Rijndael, hence invalidating any existing hardware accelerated implementations such as AES New Instructions in x86
    - CAST-256 has uniform speed across platforms, allowing for fair mining
    - CAST-256 has large S-boxes which cannot be optimized for hardware implementations, driving up the cost of specialized hardware
- The mixing round was changed to include the following operations (the order of which is dependent on the data and level of recursion):

– CAST-256 encryption.
– 64-bit multiplication.
– Data-dependent application of BLAKE, Groestl, Skein or JH.
– CAST-256 decryption.

5 Characteristics

Unprll has certain characteristics which differ from other coins:

5.1 Advantages

1. Due to the wide usability of CPUs, mining is made more accessible.
2. Mining is truly egalitarian, as single threaded performance is comparable across CPUs.
3. The method outlined is futureproof, because even if single threaded performance is improved, there's a limit upto which performance can be scaled upward, instead of outward.
4. Mining marketplaces and botnets lose effectivity, as multithreaded mining is no longer usable.

5.2 Neutral points

1. There is no way to pool compute resources, because the mining process is intrinsically sequential. Hence, only solo mining is possible. However, the stance of pool mining in a cryptocurrency is disputed.
2. As the number of miners increase, the chances of any one miner getting a block decrease, to the point where a miner considers mining ineffective. This is a common pattern seen across cryptocurrencies which is usually solved with pool mining

5.3 Disadvantages

1. Verification is O(n) in the number of iterations, leading to a potential DDoS attack on the network.
2. While mining to one wallet is effectively single threaded, mining with multiple wallets cannot be restricted.
3. If the hash function used fails to resist ASICs, CPU mining loses effectivity.

6 Reference code

The code for Unprll is based on Monero[5] (Copyright ©The Monero Project[6]). This strengthens Unprll with all privacy features in Monero and to have a proven codebase.

Unprll's code will be open source, available under the same license as Monero.

7 Conclusion

Unprll intends to supplement existing cryptocurrencies, not overthrow them. The following features described in this whitepaper are easily portable to other projects:

- The proof-of-work (albeit it would require a difficulty reset to compensate for single threading).
- RNJC (with minor tweaking to hamper hardware implementations).

8 Acknowledgements

I would like to thank @jtnw and anonymous reviewers for their contributions during the design and implementation process.

References

[1] https://cryptonote.org

[2] https://cryptonote.org/whitepaper.pdf

[3] https://www.amd.com/en/technologies/hbm

[4] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Performance Comparison of the AES Submissions. https://www.schneier.com/academic/paperfiles/paper-aes-performance.pdf, 1999.

[5] https://getmonero.org

[6] https://github.com/monero-project/monero/blob/master/LICENSE