---

lang: en

ecip: 1049

title: Change the ETC Proof of Work Algorithm to Keccak256

status: ~~Last Call~~Draft

~~review-period-end: 2020-10-09¶~~

type: Standards Track

category: Core

discussions-to: https://github.com/ethereumclassic/ECIPs/issues/~~13~~394

author: Bob Summerwill <bob@etccooperative.org>, Alexander Tsankov <alexander.tsankov@colorado.edu>

created: 2019-01-08

### Abstract

A proposal to replace the current Ethereum Classic proof of work algorithm Etchash with Keccak-256.

Motivation

* A response to the recent double-spend attacks against Ethereum Classic. Most of this hashpower was rented or came from other chains, specfically Ethereum (ETH). A seperate proof of work algorithm would encourage the development of a specialized Ethereum Classic mining community, and blunt the ability for attackers to purchase mercenary hash power on the open-market.

* As a secondary benefit, deployed smart contracts and dapps running on chain are currently able to use `keccak256()` in their code. This ECIP could open the possibility of smart contracts being able to evaluate chain state, and simplify second layer (L2) development.

Rationale

### Reason 1: Similarity to Bitcoin

The Bitcoin network currently uses the CPU-intensive SHA256 Algorithm to evaluate blocks. When Ethereum was deployed it used a different algorithm, Dagger-Hashimoto, which eventually became Ethash on 1.0 launch. Dagger-Hashimoto was explicitly designed to be memory-intensive with the goal of ASIC resistance [1]. It has been provably unsuccessful at this goal, with Ethash ASICs currently easily availalble on the market.

Keccak256 (aka SHA3) is the product of decades of research and the winner of a multi-year contest held by NIST that has rigorously verified its robustness and quality as a hashing algorithm. It is one of the only hashing algorithms besides SHA256 that is allowed for military and scientific-grade applications, and can provide sufficient hashing entropy for a proof of work system. This algorithm would position Ethereum Classic at an advantage in mission-critical blockchain applications that are required to use provably high-strength algorithms. [2]

A CPU-intensive algorithm like Keccak256 would allow both the uniqueness of a fresh PoW algorithm that has not had ASICs developed against it, while at the same time allowing for organic optimization of a dedicated and financially commited miner base, much the way Bitcoin did with its own SHA256 algorithm.

If Ethereum Classic is to succeed as a project, we need to take what we have learned from Bitcoin and move towards CPU-hard PoW algorithms.

> At first, most users would run network nodes, but as the network grows beyond a certain point, it would be left more and more to specialists with server farms of specialized hardware. - Satoshi Nakamoto (2008-11-03) [3]

Note: Please consider this is from 2008, and the Bitcoin community at that time did not differentiate between node operators and miners. I interpret "network nodes" in this quote to refer to miners, and "server farms of specialized hardware" to refer to mining farms.

### Reason 2: Value to Smart Contract Developers

In Solidity, developers have access to the `keccak256()` function, which allows a smart contract to efficiently calculate the hash of a given input. This has been used in a number of interesting projects launched on both Ethereum and Ethereum-Classic. Most Specifcally a project called 0xBitcoin [4] - which the ERC-918 spec was based on.

0xBitcoin is a security-audited [5] dapp that allows users to submit a proof of work hash directly to a smart contract running on the Ethereum blockchain. If the sent hash matches the given requirements, a token reward is trustlessly dispensed to the sender, along with the contract reevaluating difficulty parameters. This project has run successfully for over 10 months, and has minted over 3 million tokens [6].

With the direction that Ethereum Classic is taking: a focus on Layer-2 solutions and cross-chain compatibility; being able to evaluate proof of work on chain, will be tremendously valuable to developers of both smart-contracts and node software writers. This could greatly simplify interoperability.

Implementation

Work in Progress:

Example of a Smart contract hashing being able to trustlessly Keccak hash a hypothetical block header.

![example](https://i.imgur.com/xh3WgCF.png)

Here is an analysis of Monero's nonce-distribution for "cryptonight", an algorithm similar to Ethash, which also attempts to be "ASIC-Resistant" it is very clear in the picture that before the hashing algorithm is changed there is a clear nonce-pattern. This is indicative of a major failure in a hashing algorithm, and should illustrate the dangers of disregarding proper cryptographic security. Finding a hashing pattern would be far harder using a proven system like Keccak:

![example](https://i.imgur.com/vVdmzm9.jpg)

Based on analysis of the EVM architecture [here](https://cdn.discordapp.com/attachments/223675625334898688/534597157693685760/eth.jpg) there are two main pieces that need to be changed:

1. The Proof of work function needs to be replaced with Keccak256

~~1~~2. The Function that checks the nonce-header in the block needs to know to accept Keccak256 hashes as valid for a block.

![example](https://i.imgur.com/2hobqOL.png)

After doing further analysis it the best way forward to begin work is to implement this change in [Multi-Geth](https://github.com/ethoxy/multi-geth) instead of any other client. This is because Multi-geth is organized for multi-chain development, it seems to be more recently updated than classic-geth, and it is designed to be used with alternative consensus methods- which is necessary for implementing ECIP-1049.

The area where most of the changes will be in `multi-geth/consensus`

### References:

1. https://github.com/ethereum/wiki/wiki/Dagger-Hashimoto#introduction

~~1~~2. https://en.wikipedia.org/wiki/SHA-3

~~1~~3. https://satoshi.nakamotoinstitute.org/emails/cryptography/2/

~~1~~4. https://github.com/0xbitcoin/white-paper

~~1~~5. https://github.com/EthereumCommonwealth/Auditing/issues/102

~~1~~6. https://etherscan.io/address/0xb6ed7644c69416d67b522e20bc294a9a9b405b31

### Related Discussions:

1. https://github.com/ethereumclassic/ECIPs/pull/8

2. https://github.com/ethereumclassic/ECIPs/issues/13

3. https://github.com/ethereumclassic/ECIPs/issues/342

4. https://github.com/ethereumclassic/ECIPs/issues/333

5. https://github.com/ethereumclassic/ECIPs/issues/362

6. https://github.com/ethereumclassic/ECIPs/issues/382

7. https://github.com/ethereum/EIPs/issues/2951

8. https://vimeo.com/464336957

9. https://github.com/ethereumclassic/ECIPs/issues/394