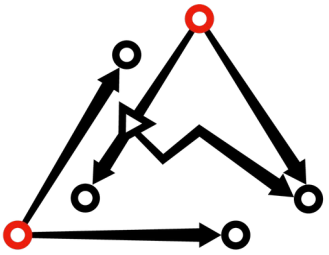


Least Authority

PRIVACY MATTERS

ProgPow Algorithm

Final Security Audit Report

Ethereum Cat Herders Ethereum Foundation and Bitfly

Report Version: 09 September 2019

Table of Contents

- Overview
  - Background
  - Project Dates
  - Review Team
- Coverage
  - Target Code
  - Areas of Concern
- Findings
  - Summary
  - Components
    - Cryptography
      - Keccak Hash Function
      - KISS Random Number Generator
      - RandMemoHash
    - Bandwidth & I/O Bounds
      - Light-Evaluation Method Mining
      - Parallelized Hypercube Architecture
    - Random Math Core
    - Independent Hardware Audit & Consultation
- Recommendations
  - Summary Table
  - Suggested Actions & Discussions
    - Suggestion 1: Scrutinize the Custom Keccak Function
    - Suggestion 2: Address the Light-Evaluation Method Mining Attack
    - Suggestion 3: Create Additional Documentation
    - Suggestion 4: Establish a Security Framework for Evaluating ASIC Resistance
    - Suggestion 5: Monitor Hardware Industry Advances
- Methodology
- About Least Authority
- Appendix A: Third Party Analysis & Resources

Overview

Background

Ethereum Cat Herders, Ethereum Foundation, and Bitfly have requested that Least Authority perform a security audit of ProgPow, a Programmatic Proof-of-Work (PoW) algorithm to replace Ethash, in order to verify the security of the algorithm and provide clear metrics about its performance.

This is part of the overall effort to examine ProgPow in order to achieve the following goals and expectations, as per the Ethereum Cat Herders:

1. "The expected effects of ProgPoW on the security of Ethereum vis-a-vis: Security of the algorithm, attack surface, cost of 51% attack, and other security risks that may result from a change from Ethash to ProgPoW.

2. ProgPoW meeting the goal of ASIC resistance: Known methods to speed up the calculation of the hash function, the length of time it would take to create a ProgPoW ASIC (if R&D begins immediately), and expected efficiency gains from the first generation of said ASICs.

3. Identify any potential advantages or disadvantages that ProgPoW would present in comparison to Ethash in terms of changes to the network, "fair mining" and evaluate any potential uneven distribution."

Project Dates

The following was the project schedule for this review and report:

- **July 17 - August 14**: Code review completed
- **August 16**: Delivery of Initial Audit Report
- **August 17 - September 6:** Feedback and consultation period
- **September 09:** Delivery of Final Audit Report

Review Team

The following Least Authority team members participated in the review and analysis of the ProgPoW algorithm, along with the preparation of this report (in alphabetical order):

- Abigail Garner - Project Manager
- Emery Hall - Security Researcher and Engineer
- Katharine Jarmul - Security Researcher and Engineer
- Hind Kurhan - Program Manager
- Ramakrishnan Muthukrishnan - Security Researcher and Engineer
- Mirco Richter - Mathematician, Security Researcher and Engineer
- Liz Steininger - CEO / Managing Director
- Dominic Tarr - Security Researcher and Engineer
- Jan Winkelmann - Security Researcher and Engineer

We also collaborated with the following people for better understanding of the hardware impact (in alphabetical order):

- Nikos Anastasiadis, FPGA / Verilog Engineer
- Bob Rao, a semiconductor technologist and retired Intel Fellow
- Piotr Steininger, software architect and GPU mining rig builder and operator

Coverage

Target Code

For this audit, we performed research, investigation, and review of the ProgPoW algorithm followed by issue reporting, along with mitigations and suggestions outlined in this report.

The following code repositories are considered in-scope for the review:

- ProgPow: https://github.com/ifdefelse/ProgPOW

Areas of Concern

Our investigation was to focus on the following areas:

- The expected effects of ProgPoW on the security of Ethereum, including but not limited to
  - The security of the algorithm,
  - The attack surface,
  - The cost of a 51% attack, and
  - Other security risks that may result from a change to ProgPoW.
- Analysis of potential advantages or disadvantages that ProgPoW would present in comparison to Ethash, including:
  - The impact on “fair mining” and potential uneven distribution of advantages,
  - Methods to impact the hash function calculation,
  - Possible changes to hash power and miner balance, and
  - Decentralization of advantages.
- Other potential effects impacting the ecosystem at large (distribution, economies of scale, cost, etc.) and other externalities of such a change.
- Anything else as identified during the initial analysis phase.

Findings

Summary

Based on our review of the ProgPow algorithm, we find that the code is accurate to its design and that it achieves its goals by more optimally utilizing GPUs than Ethash; however, we caution that future hardware advancements may potentially jeopardize this status.

Our investigation and analysis found that ProgPoW's high level design goals, summarized as "GPU-targeting and ASIC-resistant", are reasonable towards achieving its intended economic effect. We found no major issues and the design appears to function as intended, to encourage mining from diverse participants, while preventing the concentration of mining influence. Ethash, the current PoW algorithm used by the Ethereum network, already contributes to placing ASICs at a disadvantage. However, ProgPoW goes further to make the energy used per hash less divergent between a GPU and a custom ASIC. As a result, in comparison to Ethash, we find that the ProgPoW algorithm provides better overall security against recentralization which is largely based on more optimal utilization of the overall features of a GPU. By preventing ASICs from out-performing GPUs, this encourages distribution of advantages in hardware development and therefore is a likely better defense against a 51% attack.

ProgPoW's modified use of random math and parallelism, however, are approaches that have not yet been fully proven for the longer term, especially considering the fast advancements in the hardware industry. Some additional review time is suggested for specific areas of concern, though this must be balanced with the risks present at the time of investigation. Regardless of the length of any given review and analysis of ProgPoW, the possibility remains that the algorithm's new approaches may be insufficient or become obsolete over time.

Components

The following components are areas that we investigated due to their potential for attack vectors:

Cryptography

We started our analysis with a general focus on the identification of major cryptographic flaws and found nothing of particular concern in this line of investigation.

Keccak Hash Function

We examined the hash function used by ProgPoW - a non-standard instance of the Keccak function that is optimized for 32bit architectures. ProgPoW uses Keccak parameters $w=800$, $b=576$, $c=w-b=224$, and the padding is omitted.

In cryptocurrency mining, it is necessary that the hash constitutes a proof that a certain amount of computational effort was exhausted. Furthermore, it must be practically impossible to find different blocks that have identical hashes. As a result, the desired outcome is that there is no faster way to compute $H(M)$ than to evaluate H at M. Such properties are moderately well-studied for memory-hard hash functions like scrypt, but it appears that there is no existing literature available relevant to the present case. This might suggest that the topic is not considered a significant enough issue by researchers.

Nevertheless, it is still worthwhile to compare the parameter chosen for ProgPoW with that of other instances of Keccak. For collision-resistance and preimage-resistance, the most important parameter is c, resulting in security level of c/2=112 bit. The ProgPoW hash function is most closely related to the SHAKE function family, where SHAKE-128 uses $c=256$, while SHAKE-256 uses $c=512$. This implies that the

ProgPoW hash has an overall collision-resistance comparable to that of SHAKE-128. It is recommended that the Ethereum community decide whether or not such a level of security is appropriate.

One major difference between ProgPow's instance of Keccak in comparison to other instances of the hash function is the lack of padding. Since the algorithm only hashes the blockchain-block headers, seed and digest, and these collectively have the exact same length as a block in the Keccak, this is a reasonable decision. Otherwise, another Keccak-block of pure padding would have to be processed.

That being said, making a confident statement on whether this customized version fulfills expected security standards requires further scrutiny. Additional research is encouraged by cryptographers who have the relevant expertise for the specific question at hand. *See Suggestion 1*.

KISS Random Number Generator

Our review and analysis results in our agreement that the KISS random number generator is sufficient to make the sequence of math in the main loop unpredictable.

RandMemoHash

We found that an inconsistency exists, beyond the reduction in rounds, between the mention of "RandMemoHash" in the Ethash documentation and the pseudocode in the RandMemoHash specification.

Bandwidth & I/O Bounds

In addition to assessing the cryptography, we expanded our assessment to include non-cryptographic attacks, mostly by evaluating the claimed I/O/bandwidth hardness of ProgPoW - independently and in comparison to Ethash. We looked at the utilization of the DAG during mining to analyze the bandwidth hardness of ProgPoW and concluded that it is at least as secure and risks do not exceed those that exist in Ethash.

The ProgPoW inner loops that read from the DAG are pseudo-random and sequential. This pattern is unpredictable and does not provide an opportunity for unintended parallelization other than that which is intended with the ProgPoW lanes. The size of the DAG is still too large to be currently written into fast access memory, like a L3-cache or an ASIC scratchpad, without making production of the ASIC prohibitively expensive.

In Ethash, each DAG read consists of 128 bytes; in ProgPoW it is 256 bytes. Each hash calculation consists of an outer loop that runs 64 times. Each of these loops reads 16x4 32-bit values (256 bytes) from random indices calculated from a state array which is updated in every outer loop iteration. As a result, DAG reads have to occur in a sequential manner and such a random access pattern to the DAG makes successful caching strategies unlikely.

Ethash's moderately good level of ASIC-resistance via memory-hardness is therefore inherited by ProgPoW and, thus, all previous arguments on I/O-bounds are still valid. Moreover, there appears to be better utilization of the available memory bandwidth with 256-byte reads.

We also looked at the Ethash specific dataset called the cache and considered whether this cache itself could be generated on-the-fly. If an even smaller dataset is written into ultra-fast access memory, like SRAM, then every required DAG entry might be generated from this set on the fly. To confirm this, we wrote a proof of concept program that passes tests and looked into finding a time/memory trade-off for mining from this small but fast memory. We concluded that this approach is not necessary, since with up-to-date hardware ASICs, the entire cache might be written into ultra-fast memory very close to the computing units of an ASIC, if the goal is to execute light-evaluation mining.

Light-Evaluation Method Mining

Both Ethash and ProgPow have strong bandwidth requirements, although this is based on the assumption that the DAG is used during mining and that DAG nodes are not generated on-the-fly using the hashimoto-light function. The designers of Ethash are aware of this, which appears to inform their reasoning for initially using a cache that is not too small.

From the original ***Ethash-Design-Rationale.md*** file:

> "A 16 MB cache was chosen because a smaller cache would allow for an ASIC to be produced far too easily using the light-evaluation method. [..]".

During our discussions with Bob Rao, we learned that on-die scratchpad memory of around 100MB is possible in ASICs, that we can fetch at least 128 bytes during a single read, and that such a read might have a latency in the range of one to some tens of cycles.

Under those circumstances, an advanced ASIC might circumvent the DAG altogether to generate all relevant DAG nodes on-the-fly from cache using a specialized hardware implementation of the `calc_dataset_item()` function. To be more precise, in Ethash or ProgPow, such a miner might replace every dag-read `dag[i]` with a call to the function `calc_dataset_item(cache, i)` where the latter is implemented as an appropriate ASIC that stores cache in an ultra-fast on-die SRAM storage.

This line of reasoning might be called a "light-evaluation attack" and it would circumvent every assumption related to the bandwidth bounds, because no DAG is used in the first place.

To analyse the possibility of such an attack, we compare the throughput and energy consumption of two things:

1. Reading entries from the Ethash-DAG, and
2. Generating DAG entries on-the-fly using an ASIC version of `calc_dataset_item()`.

According to this publication (table 2), Keccak might be implemented on an ASIC with a latency of 25 cycles, which implies that something similar might also hold true for `keccak_f800_progpow` Moreover, both the FNV function, as well as the modulus operation, might be implemented with very small latency on an ASIC. This is only a rough estimate but it implies that the expected latency of an ASIC version of `calc_dataset_item()` could be very close to the number:

$$\texttt{DATASET\_PARENTS} * k1,$$

where $k1$ is the latency it takes to make a read to the on-die `cache[]` array. The latter of which is currently a small number, which might converge to one in the near future. As a result, generating any required DAG entry on-the-fly could have a latency as low as ~300 cycles. This has to be compared with the expected latency of a single `dag[.]` read of equal length.

A standard throughput computation as provided in Efficient hardware implementations of high throughput SHA-3 candidates keccak, luffa and blue midnight wish for single- and multi-message hashing (section 3):

$$\text{throughput} = (\text{blocksize})/(\text{cycles}) * \text{frequency},$$

implies that these 300 cycles might result in an even higher overall throughput compared to a GPU read to `dag[.]`, since an ASIC can be clocked at a higher frequency than a GPU.

Taking this into consideration, in the future, the light-evaluation method might generate DAG entries on-the-fly in a way that is as fast as a lookup to the original DAG RAM. Since a DAG is not required, all bandwidth bound assumptions disappear.

However, the light-evaluation method is economically favorable in mining only if the energy consumption of a call to the hypothetical ASIC function `calc_dataset_item_ASIC()`is not greater than a simple lookup (of a DAG-node) in RAM, in addition to the overall energy consumption of the RAM during runtime. We were not able to estimate these numbers properly due to time limitations and further considerations might be required.

Following our discussions with Bob Rao, it was determined that, in general, the energy expended/bit to access DRAM is ~3pJ/bit, but when the memory access is on-chip, it decreases to 0.3pJ/bit, which is a 10x improvement. This suggests that sufficiently parallelized light evaluation mining might give high rates of hashes/Joule.

To further analyse this attack, we assume that ASIC designers are able to build hardware that stores the cache on ulta-fast SRAM inside an ASIC, such that a function `calc_dataset_item_ASIC(.)`exists and is able to compute a DAG entry `dag[.]` as fast as a GPU needs to lookup the same DAG entry `dag[]` in its RAM. Moreover, we assume that energy consumption of function `calc_dataset_item_ASIC(.)`on the ASIC and `read(dag[.])` on the GPU/RAM are of similar magnitude.

Then in Ethash, the computation could essentially be doubled by running two of these ASICs concurrently in an attempt to parallelize the following loop in the hashimoto function:

```
for j in range(MIX_BYTES / HASH_BYTES):
  newdata.extend(dataset_lookup(p + j))
  mix = map(fnv, mix, newdata)
```

Since $\text{MIX\_BYTES} / \text{HASH\_BYTES} = 2$, we are able to generate the array `newdata[]` in parallel using two independent hardware instances of the function `calc_dataset_item_ASIC()`as replacements for `dataset_lookup()`. This means that we calculate `newdata[0]=calc_dataset_item_ASIC(p)`and `newdata[1]=calc_dataset_item_ASIC(p+1)`in parallel. However, this does not give any advantage as a GPU is able to read a similar amount of data in parallel from the DAG.

Similarly, the light-evaluation method can not be used for massive parallelization in ProgPoW, which is essentially due to the unpredictability of the DAG access pattern.

In conclusion, the light-evaluation method attack might become a threat, only if on-die ASIC SRAM evolves to the point where sizes of approximately 100MB can be accessed in very few cycles and an ASIC can be built for the function `calc_dataset_item()`that consumes no more energy per call than a read to the DAG from a GPU. Furthermore, the circumstances in ProgPoW are much more favorable due to the additional random math core. Even in the event that a hypothetical function `calc_dataset_item_ASIC()`is very efficient, the random math core likely prohibits the build of a light-evaluation based ASIC. *See* *Suggestion 2*.

Parallelized Hypercube Architecture

Both Ethash and ProgPoW perform a random memory lookup followed by a hash/mix operation. In a GPU or current ASIC, each unit has a processor and enough memory to hold the entire DAG, while the limiting factor is the memory bandwidth. However, if the memory was split into many smaller blocks and in separate chips, the total memory bandwidth could be increased linearly with the number of chips. To take advantage of this, instead of performing the mix and then performing the next lookup from a single chip and memory bank pair, the mix would be sent to the chip closest to that section of memory. Sending this to another processor incurs overhead, but in turn frees the current chip to do another calculation.

A much larger mix state would not make it feasible to send it around to different processors and save time, but this may also make it harder to fit inside the L1 cache. Ethash uses a 128 byte mix (the same size as a memory lookup) but ProgPoW has a substantially larger mix (2048 bytes) so it may be less susceptible as a result. Although this sort of architecture is common in supercomputing, it still contains many challenges and it is questionable whether it is a realistic approach in the near future.

Random Math Core

The major key difference between Ethash and ProgPoW is the random math core. As a result, we investigated the effectiveness of this variation. The rationale of extending Ethash into a programmable PoW, by integrating a random math core, is to utilize current GPUs as much as possible.

The `math()` and `merge()`routines take a random selector and run the same series of instructions in all 16 lanes but with different data, keeping the lanes busy. The overall design appears to be tailored for a GPU-like architecture where there are many functional units that all run the same instructions but can operate on different sets of data. We find the overall design to be sound. However, the `merge()` and `math()` programs do not use the common "multiply and add" instruction available on most GPUs. Instead, the math instructions use integer operations, but do not use floating-point arithmetic. This issue has been raised by others on various online forums and the reason mentioned was that GPUs are not guaranteed to give the same results as CPU even though they are IEEE 754 compliant. Like addition, floating-point operations are not associative (i.e. $a + (b + c)$ is not equal to $(a + b) + c$), so the order of the operations have an effect on the output.

The randomly generated program remains constant for all blocks in one `PROGPOW_PERIOD`which is currently defined to be 10 in version 0.9.3. We observe two different effects this might have on the network. First, the resulting variation in hash rate between periods might create fluctuations in the difficulty function. However, shortening the period will smooth this out. The second aspect is further increased resistance to FPGAs, as building and loading a bitstream in such a short period (~2 minutes) would make it impractical.

Overall, we found that the additional use of random math sequences extends the ASIC resistance of Ethash substantially by utilizing not only the DAG, but more of the GPU's potential. ProgPoW's ASIC resistance is therefore not only based on bandwidth, but on targeting GPUs specifically.

Independent Hardware Audit & Consultation

In a separate effort from the Least Authority audit, Bob Rao, a semiconductor technologist and retired Intel Fellow, is currently in the process of performing an independent audit from the perspective of the hardware areas of concern.

Throughout the duration of our audit, we spoke to Bob Rao twice and shared some email correspondence about our lines of investigation and analysis from the perspective of the software concerns. These conversations resulted in notable discussions and analysis points that we have documented.

What we learned is that external memory access is a lot more power consuming than computation done with data within the chip. In addition, we learned about SRAM densities in the die and that one cannot infinitely increase the amount of SRAM due to the following reasons:

1. The yield of operative chips decreases exponentially as the chip area increases;
2. Access latency grows with the size of the SRAM

It is important to note that, at the time of writing this *Initial Audit Report*, we had only spoken to Bob Rao, but have not reviewed a formal report or published documentation from him. As a result, these points should be reconsidered upon the sharing and review of his independent audit report.

We also spoke to an independent FPGA Designer, Nikos Anastasiadis, for additional feedback on the role of FPGAs in the GPU versus ASIC assessment. Based on those discussions, the conclusion that an ASIC design would look similar to a GPU design was reinforced.

Recommendations

Summary Table

SUGGESTION	STATUS
Suggestion 1: Scrutinize the Custom Keccak Function	Reported
Suggestion 2: Address the Light-Evaluation Method Mining Attack	Reported
Suggestion 3: Create Additional Documentation	Reported
Suggestion 4: Establish a Security Framework for Evaluating ASIC Resistance	Reported
Suggestion 5: Monitor Hardware Industry Advances	Reported

Suggested Actions & Discussions

The following is a list of suggested areas to be further explored, considered and discussed by the involved teams and the broader community.

Suggestion 1: Scrutinize the Custom Keccak Function

Synopsis

The Keccak function variant in ProgPoW does not use padding. Intuitively, this is a safe change, especially because the hashed data is exactly one Keccak-block long. However, while rounds and the parameters b, c, and r are configurable, the padding used in Keccak is fixed to multi-rate padding. This technically means that the hash is not a Keccak instance.

Mitigation

Professionals with experience researching and investigating Keccak should further explore the hash function.

Outcome

A deeper look at the custom Keccak function could elicit previously unidentified security risks.

Suggestion 2: Address the Light-Evaluation Method Mining Attack

Synopsis

Although no immediate changes are required it is concluded that efficient light-evaluation attacks may become possible within a few years. This is also an issue that applies to Ethash.

Mitigation

The threat is easily mitigated by changing the constant `DATASET_PARENTS`to a higher value like 512. Although this would not change the size of the DAG or any of the benchmarks and analytics of the mining process using DAG lookups, it would change the amount of cycles required for verification.

We recommend increasing the constant `DATASET_PARENTS` value on Ethash and ProgPoW, before such a threat becomes real. For details on the related hardware advancements, please see Bob Rao's corresponding audit report.

Outcome

The Light-Evaluation Method Mining Attack would be prevented and this would no longer be a potential attack vector benefiting a subset of miners with access to the advanced hardware.

Suggestion 3: Create Additional Documentation

Synopsis

Upon reviewing the ProgPoW documentation, it appears to be missing key details. Improving the current documentation could be useful for future review and assessments.

Mitigation

It would be helpful if more details about the importance of entropy conservation and the reasons for the selected atomic math functions were specifically addressed in the documentation. Also, collecting the documentation into one resource would facilitate better use and access of the information it contains.

Outcome

By improving the documentation, it will make it easier for other community members to better understand how ProgPoW works. This is advantageous for not only contributions, but also facilitates future security reviews and feedback from the community which is another aspect of decentralization .

Suggestion 4: Establish a Security Framework for Evaluating ASIC Resistance

Synopsis

Memory hard algorithms are still relatively new and require further investigation and research. Furthermore, cryptography requires time to mature. As a result, a framework for evaluating the use of random computations or memory lookups does not currently exist as an additional security benchmark for ASICs. Prudence requires the assumption that there is much to learn about ASIC resistance as new architecture is developed to enable the technology and a reliable model would aid in that effort.

Mitigation

Explore creating a security framework for evaluating ASIC resistance, particularly if mining continues to be a growing industry.

Outcome

By organizing a group within the community to create a security framework for evaluating ASIC resistance, the status of ProgPoW can be better monitored over time.

Suggestion 5: Monitor Hardware Industry Advances

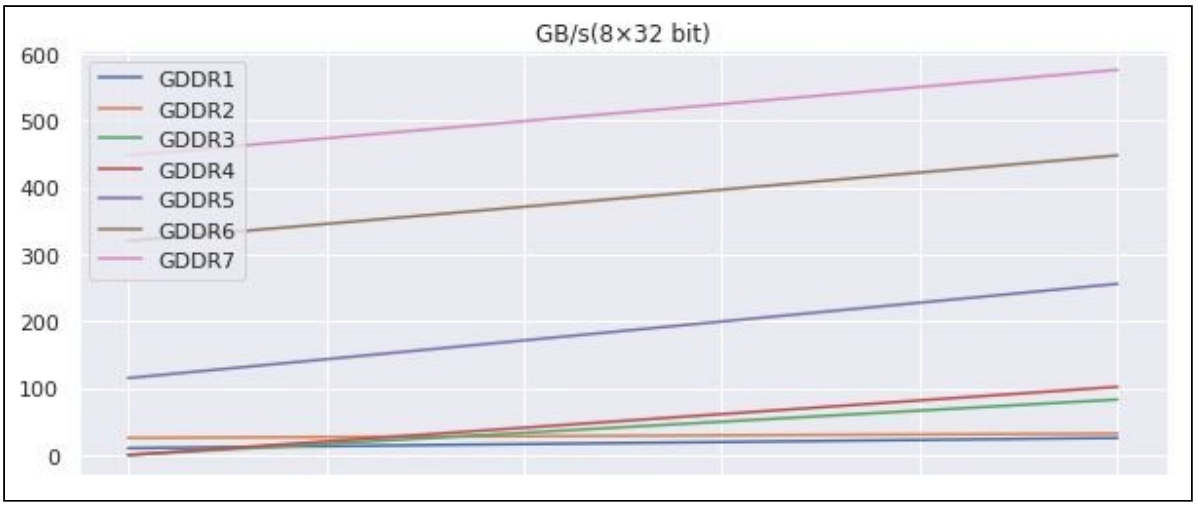
Synopsis

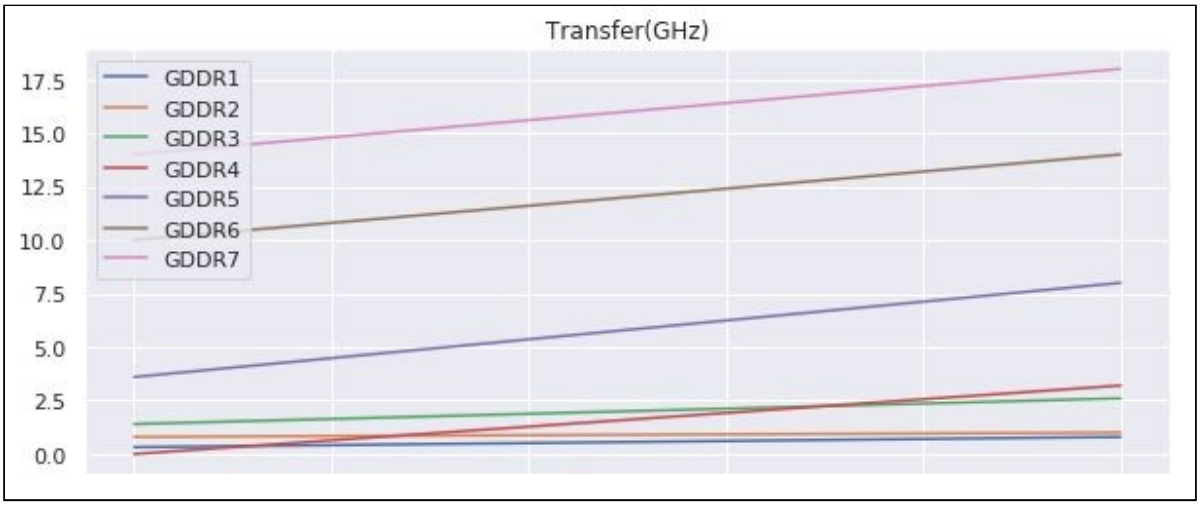
Mining and gaming are not the only industries that utilize GPUs and have incentives to rapidly improve the state-of-the-art in hardware. The field of Machine Learning has seen increased research and development investment and also utilizes GPU hardware. Companies such as Google and Microsoft have built

specialized hardware (such as Google TPU and Microsoft Catapult) and numerous governments have built hardware research teams in what has been referred to as the "AI Arms Race". These companies have focused on similar problems that ProgPoW has targeted -- namely increased bandwidth, larger caching and memory, as well as energy efficiency. It is likely that hardware targeting machine learning is also useful for ProgPow mining.

Specialized machine learning hardware aimed at increasing bandwidth to computation units (either ALU, GPU, TPU or FPGAs) could be repurposed for ProgPoW mining. This hardware may not be available to the average consumer depending on the development costs (i.e. Google TPUs are currently only available to Google employees, large university researchers, and consumers via Google Cloud).

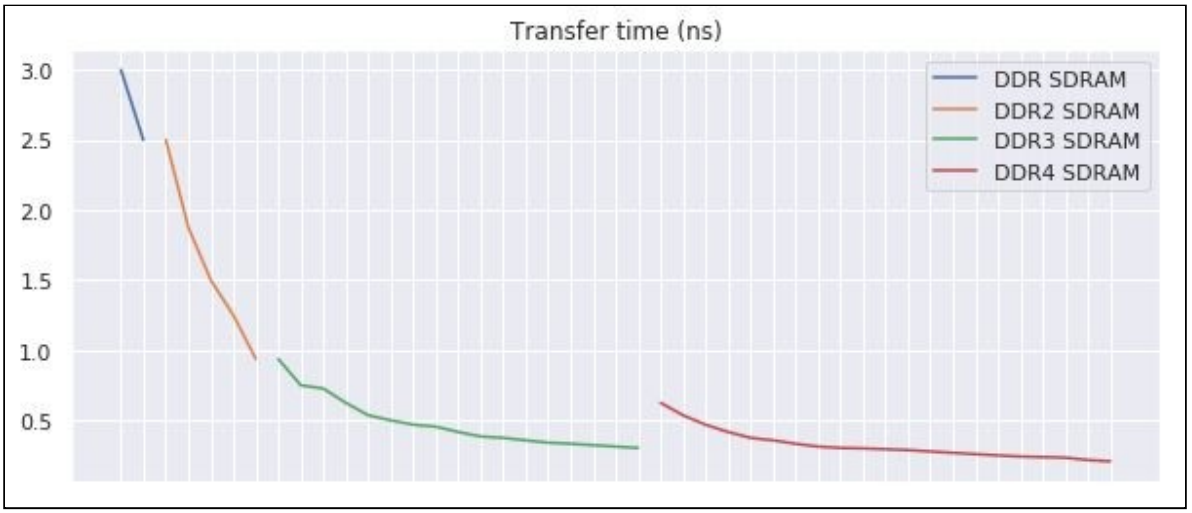
As shown in the charts below, GDDR speed-ups can be difficult to predict between new generations.





Source: https://de.wikipedia.org/wiki/Graphics_Double_Data_Rate

This observation also applies to increases in data rate for SDRAM.



Source: https://en.wikipedia.org/wiki/DDR_SDRAM

Due to the inability to adequately predict advances in memory-bound computation as well as the active research and development within the industry, it is unlikely ProgPoW can properly anticipate potential changes and availability of new hardware which may result in uneven performance amongst miners.

Mitigation

Similar to the inability to predict whether ProgPow could be built into a viable ASIC, it is impossible to accurately predict the ability to use specialized deep learning hardware to give miners an unfair advantage. Outside of specialized hardware, the current GPU producers have incentives to continue to release more powerful equipment for deep learning on a regular basis. Because this equipment is often quite expensive at first, this could create an unequally distributed market for miners, especially benefiting those with access to unused or idle deep learning rigs. Since it is rare that GPUs at a deep learning startup or company are 100% utilized, it has led to several startups and discussions regarding selling GPU cycles to other researchers. It is unclear if the value of Ethereum will increase enough to make this a competitive market between AI researchers, gamers, and ProgPoW miners.

It is recommended that further analysis be conducted on state-of-the-art deep learning equipment (Google TPU, Microsoft Catapult, Nvidia latest devices) by doing the following:

1. Monitor and test new hardware as they become available for potential advantages.
2. Determine parameters that help regularize performance across different hardware architectures (i.e. limiting the impact higher performance GPUs might have versus older versions). This would help create a more level playing field between different generations and memory capacities for different GPU/TPU/NPUs.

This could be done as a follow-up analysis by a hardware expert familiar with these latest devices or through a community research project. In general, this area of advancement should be monitored for significant changes in the coming years.

Outcome

By monitoring and investigating hardware advancements from other fields of application, challenges to ProgPoW's design, and therefore security, can be identified and addressed sooner rather than later.

Methodology

We work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we used in our security audit process for this review.

Manual Code Review

In manually reviewing all of the code, we looked for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watched for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We looked for potential vulnerabilities that were discovered there or missed. We brainstormed the ASIC performance threat model, along with other potential threats to the effectiveness of the algorithm and various attack surfaces. We hypothesize what vulnerabilities may be present, creating Issue or Suggestion entries for each, and follow our Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation or mitigation. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or through other tests. After this, we analyze the feasibility of an attack in a live system.

Responsible Disclosure

Our report and any details about our findings and suggestions are to be shared with the Ethereum core developer community, along with our noted project clients. We hope to work with the community to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for resolution that balances the impact on the users and the needs of the Ethereum core development team.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. Although we are a small team, we believe that we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Appendix A: Third Party Analysis & Resources

As part of this review, the Least Authority team researched the previous and ongoing reviews and comments by other parties in the broader community.

The following is a list of the third party assessments that we looked at

1. Understanding ProgPoW, 2018, Medium post by IfDefElse: https://medium.com/@ifdefelse/understanding-progpow-performance-and-tuning-d72713898db3 The agreed-on standpoint of ProgPoW being GPU optimal (more or less) can be distilled from here.
2. ProgPow ASIC possibilities evaluated by 2 experts, 2019, IfDefElse GitHub: https://github.com/ifdefelse/ProgPOW/issues/24
3. ProgPoW: Progress Update #1, 2019, IfDefElse Medium post: https://medium.com/@ifdefelse/progpow-progress-da5bb31a651b
4. Ethash Spec, 2018 (Revision 23): https://github.com/ethereum/wiki/wiki/Ethash
5. Analysis of Ethash by Least Authority, 2015, Least Authority GitHub: https://github.com/LeastAuthority/ethereum-analyses/blob/master/PoW.md In 2015, Least Authority performed a review of Ethash. This is the currently used Proof of Work algorithm.
6. Open Chip Design for 1% cost/power increase, Linzhi ASICs Medium post: https://medium.com/@Linzhi/eip-1057-progpow-open-chip-design-for-only-1-cost-power-increase-eip-1057-progpow-d106d9baa6eb
7. Sarah Osbourne Response to the Linzhi ASIC post: https://medium.com/@profheisenberg/everybody-knows-alus-are-relatively-tiny-circuits-a589d2de4cce
8. Security Audit of Monero RandomX, 2019, Quarkslab: https://blog.quarkslab.com/security-audit-of-monero-randomx.html
9. RandomX Security Assessment, 2019, Trail of Bits published report: https://github.com/trailofbits/publications/blob/master/reviews/arweave-randomx.pdf
10. Zcash Foundation Grant proposal by IfDefElse: https://github.com/ZcashFoundation/GrantProposals-2018Q2/issues/15
11. Zcash Foundation Grant proposal by Solar Designer: https://github.com/ZcashFoundation/GrantProposals-2018Q2/issues/25 and https://github.com/ZcashFoundation/GrantProposals-2018Q2/issues/25#issue-324037312
12. Inside the New Mining Technology That Will Redefine the Industry: https://blog.usejournal.com/inside-the-new-crypto-mining-technology-that-will-redefine-the-industry-196529547c88?gi=788f2c8ac921