LEDGER
ledgerjournal.org

ISSN 2379-5980 (online)
associated article DOI
10.5915/LEDGER.2017.48

Equihash: Asymmetric Proof-of-Work Based on the Generalized Birthday Problem: Open Review

Authors: Alex Biryukov,[†] Dmitry Khovratovich,[‡]

Reviewers: Reviewer A, Reviewer B

Abstract. The final version of the paper "Equihash: Asymmetric Proof-of-Work Based on the Generalized Birthday Problem" can be found in Ledger Vol. 2 (2017) 1-30, DOI 10.5915/LEDGER.2017.48. There were two reviewers who responded, neither of whom have requested to waive their anonymity at present, and are thus listed as A and B. After initial review (1A), the author submitted a revised submission and responses (1B). The revised submission was reviewed once again by reviewers A and B, who determined that the author had adequately and substantively addressed their concerns, thus completing the peer-review process. Authors' responses in are in bullet form.

1A. Review

Reviewer A:

The paper proposes a proof-of-work (PoW) scheme based on a problem called the generalized birthday problem. The primary goal is to develop a PoW scheme for cryptocurrency with several properties:

1. (tuneable costs) the cost of producing a PoW can be adjusted/controlled,

2. (equitable costs) the amortized cost of producing a PoWs on an ASIC is not much lower than the cost of producing a PoW on standard computer architectures (e.g., desktops), and

3. (extreme asymmetry) the cost of verifying a proof-of-work should be as inexpensive as possible.

The problem is well-motivated. For example, Hashcash, the PoW used in Bitcoin, satisfies properties 1 and 3 but not 2. The (undesirable) result is that a small number of miners (with

[†]A. Biryukov (alex.biryukov@uni.lu) is a full professor of Cryptography and Information Security in the Computer Science and Communications Research Unit at the University of Luxembourg: 16B1873NxbzzX2xNhW2GzYbCBJtjxzF23j
[‡]D. Khovratovich is a post-doctoral researcher at the CryptoLUX cryptography research group at the University of Luxembourg: 1PbreZUDQjpVWFLBSZp3aimaqSyoiLa5eT

ASICs) control most of the hash power in the Bitcoin ecosystem. PoWs based on memory hard functions (*e.g.*, SCRYPT) potentially satisfy properties 1 and 2, but the cost of verifying a PoW is undesirably high.

The authors present a PoW candidate based on the Generalized Birthday Paradox and provide some arguments that it satisfies all three properties. One of the primary technical tools they use is an idea called `algorithm binding.' The goal is to tie the PoW solution to a particular execution of Wagner's algorithm for the generalized birthday paradox problem so that the amortized cost of computing multiple PoW solutions is equal to the cost of computing one PoW solution.

The equihash scheme relies on several key assumptions: (hardware/memory) it is not possible to exceed memory bandwidth 1 TB/s on chip, (parallel sorting) *any* parallel sorting algorithm requires high memory bandwidth and (no shortcuts) Wagner's algorithm requires sorting. These assumptions need to be acknowledged/discussed earlier in the paper. Furthermore, as these assumptions are crucial the discussion around these points needs to be expanded. In particular, the paper only cites [33] and [40] when discussing parallel/sequential sorting. Both papers are several (2012) years old so I would not be shocked if more recent GPUs/ASICs have better performance.

Technical Comments:

Proposition 1 only tells us how many solutions we get on average using Algorithm 1 with a specific memory size. It would be helpful to (at least briefly) comment on variance (*e.g.*, the proposition statement does not rule out the possibility that we get 2million solutions w.p. 10^{-6} and 0 solutions w.p 1-10^{-6}). Also during the first step we actually expect (N \choose 2)*2^{-n/(k+1)} = 2^{n/(k+1)+1} - 1 = N-1 entries for the second table. For the third table it will be N-2+o(1) etc... I think you will still end up with (almost) two solutions in expectation, but the exact formula will be slightly more complicated.

The notation in II.A (Steep time-space tradeoffs) needs to be clarified (*e.g.*, I don't understand the term max_{A_R} M(A_R) as M(A_R) is never defined. Does A_R denote the standard implementation?). How do M_0 and M relate to each other? Does C_R(q) compare the performance of a fixed algorithm A_R as memory varies or are we comparing the reference distribution with the best known algorithms with memory M/q?

Proposition 3 suggests that the amortized cost of producing a PoW solution drops dramatically as memory increases. Thus, an adversary with less memory will pay a steep cost. However, I am not convinced that this is actually a positive result for Equihash as it suggests that the only effective way to mine is to have the most memory (*e.g.*, miners may prefer to deviate from algorithm 2 to produce the PoW). It seems like we also want the property that we cannot effectively make use of memory M > M_0, where M_0 is the memory used by the reference implementation. Furthermore, I don't think that the ``algorithm-binding technique'' fixes this particular concern.

Similarly, proposition 7 seems to indicate that the scheme is broken in theoretical sense (*e.g.*, on an parallel ASIC AT costs can be decreased dramatically). In particular, there is an attack with parallelism p that reduces the time to find an algorithm-bound PoW solution by (roughly) p. The (potentially) saving factor is that the required memory bandwidth also grows by a factor of p. Thus, this particular attack may be unrealistic in practice (assuming that no ASIC/memory chip can be developed that accommodating). Proposition 7 does leave open the possibility of other (yet unknown) practical parallel attacks.

Minor:

Footnote 1: "are now integrated"

The authors only cite their own paper [15] during their discussion of time-space tradeoffs of MHFs in the introduction. See High Parallel Complexity Graphs and Memory-Hard Functions. Alwen, J. and Serbinenko, V. STOC 2015.

(More recent)

Balloon Hashing: a Provably Memory-Hard Function with a Data-Independent Access Pattern. Boneh Corrigan-Gibbs, Schechter. https://eprint.iacr.org/2016/027.pdf

Efficiently Computing Data Independent Memory Hard Functions. Alwen, J. and Blocki, J. CRYPTO 2016.

Reviewer B:

On the whole I think this submission merits publication at this venue. It is relatively well written and makes a nice contribution to the area of ASIC resistant computation which is itself undoubtedly a well motivated topic of research in the field of cryptocurrencies.

I particularly like the idea of using the generalized birthday bound as a basses for memory-hard problems. From a theoretical perspective the problem has been previously studied in various forms which provides us with various tools (algorithms) and related lower-bounds for reasoning about the construction. More practically it is simple to describe, understand and implement and the resulting asymmetry between prover & verifier are extreme. It also seems to lend itself well to tuning various lower-bounds on the resources required to produce proofs. It is also very helpful that that an implementation is available.

However I do have two concerns with the current writeup and I **strongly** suggest they should be addressed before the final version is submitted for publication.

1) The writeup is quite unclear about what is being conjectured vs. what is being proven when it comes to security. It would benefit greatly from a paragraph summarizing what is know, what is shown and what remains open/is being conjectured.

In particular my understanding is that in order for the construction to be a good PoW according to the authors own criterion it must be that no algorithm solves the "algorithm-bound" version of the k-XOR problem significantly better then the improved Wagner's algorithm and that no better optimizations exists for Wagner's algorithm then the ones listen in the writeup. Only then do we get the guarantee that the construction is "Optimization-free". In other words, only then do get the guarantee that an adversary has no advantage over the honest parties. In my opinion this is an absolutely crucial feature of any PoW construction. Why this should be true for the proposed construction should be made much clearer given it's importance.

Indeed the closest I could find for any such justification may be the sentence "Therefore, if we pre-fix the positions where 2^l-XORs have zero bits, we bind the user to a particular algorithm flow." But this sentence makes little sense to me (which is why I think it needs to be either much expanded upon, or else out right removed). On a high level, my understanding of the authors intentions here are based on the observation that Wagner's algorithm produces a particular subset of possible solutions to the k-XOR problem so the authors modify the problem to only accept solutions from this subset. However to then make the leap that this means _any_ adversary must necessarily use Wagner's, or even just a "Wagner-like" algorithm is a very big leap. In particular I think that A) "algorithm flow" or "wagner like" or similar terms should be made much more clear and precise and B) this conjecture upon which security is based should be made very explicit and highlighted. After all it is an important starting point for future work wishing to confirm or refute the security of the protocol proposed in this work.

To be clear, I am by no means asserting there to be a problem with the conjecture, or even that there is somehow good reason to believe it is false. Simply that if such a conjecture is needed for full security then it should be stated both explicitly and precisely. (Though I suspect making it precise may turn out to be not be that easy... *e.g.* is it clear that no third type of optimization, other than the two listed at the top of page 7, can not exist for Wagner's algorithm? Put differently how can one characterize the set of algorithms for which security is actually shown in the submission?) Any evidence why such a conjecture should hold would also be greatly appreciated and significantly strengthen this work.

2) The introduction repeatedly confuses memory-bound and memory-hard functions. These are not the same and using the terms and corresponding results interchangeably should definitely be avoided. Memory-bound means many cache misses are needed to compute the function. "Memory-hard" means a lot of memory is needed to compute the function. While the later may (at least in some sense) imply the former the converse is not true. For example I strongly disagree with the discussion concerning [22]. [22] does the same (*i.e.* uses the same model, security definition and a closely related proof) as [20] and has the same intuitive goal and motivation as [6, 20]. Yet in the submission, [22] is described as being "memory-hard" while [6, 22] are said to be "memory-intensive" (as an aside: why introduce a new term here? why not use the existing one used in those and other works: "memory-bound"?). However even a brief inspection of [22] and [20] show that they are actually **very** similar results. Indeed [22] is formulated explicitly as improving on the construction of [20]. More generally to see the difference between memory-bound and memory-hard consider a machine with say 1MB cache, doing random pointer jumping in a 2MB array of incompressible bits (*i.e.* essentially

the construction of [20] and [22]). This is a memory-bound function as computing it (with any algorithm) results in many cache misses (as proved in [22, 20]). However it is not memory-hard in that it doesn't require a lot of memory (only 2MB). To be clear: the goal of [22] (and [6, 22]) was **not** that an adversary requires a lot of memory. just that they require a lot of memory accesses. But this can already be achieved with not very much more memory than cache. Indeed, this is why increasing the difficulty parameter for such memory-bound function constructions only increases the number of pointer jumps, but not the size of the array in memory.

Other small comments

- Abstract: "The proof-of-work" -> "Proofs-of-work"

- Page 1: Botnets seem a bad motivation for this since they consist of the same type of hardware as honest users. given that botnets are used for other intensive computation (such as bitcoin mining) it would seem that hiding intensive resource usage from an infected hosts owner is not too great a problem (especially given the extreme under-usage of the hardware in most home PCs.)... ASIC/FPGA resistance is a far better motivation for this submission.

- Page 2: "is expected" -> "we expect" Also [23] has been implemented and is available as part of the github project working towards a full spacemint implementation. If the submission wants claim that [23] is not practical it needs to explain what is wrong with the current implementation.

- Page 2: It would be good to explain why not being amortization-free should be considered a problem for the Proof-of-Space protocols in [23] given that [37] uses them in precisely one of the main applications motivating the PoW in the submission: namely a cryptocurrency. Also I don't understand what is meant by "the computational penalties are guaranteed only after the memory reduction by a logarithmic factor". This should be made clearer and it should also be clarified why the authors believe their results address this problem. (If the statement refers to what the authors in [23] were able to prove in terms of a security statement then I feel this to be rather misleading criticism of [23]. After all 1) no "attack" on [37] is known to exist between the gap of what is proven and what one would ideally desire and 2) I am unclear on why the submission should have a significantly better state of affairs in terms of actual security guarantees. As discussed above, my understanding is that security of the submission is based on a strong lowerbound conjecture (not unlike [23] which also relies on a conjecture). Given the reliance on, a priori incomparable, conjectures criticizing exact security seems... a bit misleading.)

- Page 3: in the numerator in the equation "M/q" -> "M_S0/q"

- The authors highlight a useful property called "Progress-free Process" but then never refer to it again. It might be nice to add in a comment somewhere high lighting why the submission enjoys this property (or at least that it does).

- I'm slightly unclear as to why the authors believe that a bitcoin like PoW but using say Argon2d (being faster then scrypt to avoid the Litecoin problem) would not constitute a PoW according to their criterion. For example such a protocol exhibits the required asymmetry: The verifier only performs a single evaluation of the hash function compared to the exponentially many evaluations by the prover. In fact such a protocol would also be progress-free, allow for tuning various resource constraints and lend itself well to implementation. I doubt I will be the only one with such thoughts so I think it would help if the authors address them.

1B. Authors' Response

Reviewer A:

The paper proposes a proof-of-work (PoW) scheme based on a problem called the generalized birthday problem. The primary goal is to develop a PoW scheme for cryptocurrency with several properties:

1. (tuneable costs) the cost of producing a PoW can be adjusted/controlled,

2. (equitable costs) the amortized cost of producing a PoWs on an ASIC is not much lower than the cost of producing a PoW on standard computer architectures (*e.g.*, desktops), and

3. (extreme asymmetry) the cost of verifying a proof-of-work should be as inexpensive as possible.

The problem is well-motivated. For example, Hashcash, the PoW used in Bitcoin, satisfies properties 1 and 3 but not 2. The (undesirable) result is that a small number of miners (with ASICs) control most of the hash power in the Bitcoin ecosystem. PoWs based on memory hard functions (*e.g.*, SCRYPT) potentially satisfy properties 1 and 2, but the cost of verifying a PoW is undesirably high.

The authors present a PoW candidate based on the Generalized Birthday Paradox and provide some arguments that it satisfies all three properties. One of the primary technical tools they use is an idea called `algorithm binding.' The goal is to tie the PoW solution to a particular execution of Wagner's algorithm for the generalized birthday paradox problem so that the amortized cost of computing multiple PoW solutions is equal to the cost of computing one PoW solution.

The equihash scheme relies on several key assumptions: (hardware/memory) it is not possible to exceed memory bandwidth 1 TB/s on chip, (parallel sorting) *any* parallel sorting algorithm requires high memory bandwidth and (no shortcuts) Wagner's algorithm requires sorting. These assumptions need to be acknowledged/discused earlier in the paper. Furthermore, as these assumptions are crucial, the discussion around these points needs to be expanded. In particular, the paper only cites [33] and [40] when discussing parallel/sequential sorting. Both papers are several (2012) years old so I would not be shocked if more recent GPUs/ASICs have better performance.

Technical Comments:

Proposition 1 only tells us how many solutions we get on average using Algorithm 1 with a specific memory size. It would be helpful to (at least briefly) comment on variance (*e.g.*, the proposition statement does not rule out the possibility that we get 2million solutions w.p. 10^{-6} and 0 solutions w.p 1-10^{-6}). Also during the first step we actually expect (N \choose 2)*2^{-n/(k+1)} = 2^{n/(k+1)+1} - 1 = N-1 entries for the second table. For the third table it will be N-2+o(1) etc... I think you will still end up with (almost) two solutions in expectation, but the exact formula will be slightly more complicated.

- We have commented on the variance after Proposition 1.

The notation in II.A (Steep time-space tradeoffs) needs to be clarified (*e.g.*, I don't understand the term max_{A_R} M(A_R) as M(A_R) is never defined. Does A_R denote the standard implementation?). How do M_0 and M relate to each other? Does C_R(q) compare the performance of a fixed algorithm A_R as memory varies or are we comparing the reference distribution with the best known algorithms with memory M/q?

- We have clarified this paragraph.

Proposition 3 suggests that the amortized cost of producing a PoW solution drops dramatically as memory increases. Thus, an adversary with less memory will pay a steep cost. However, I am not convinced that this is actually a positive result for Equihash as it suggests that the only effective way to mine is to have the most memory (*e.g.*, miners may prefer to deviate from algorithm 2 to produce the PoW). It seems like we also want the property that we cannot effectively make use of memory M > M_0, where M_0 is the memory used by the reference implementation. Furthermore, I don't think that the ``algorithm-binding technique" fixes this particular concern.

- We have discussed this in a new subsection V.D. Unfortunately there is no way to prevent miners from slight optimizations in either time or memory from the reference implementation, but we can prove that these optimizations are limited.

Similarly, proposition 7 seems to indicate that the scheme is broken in theoretical sense (*e.g.*, on an parallel ASIC AT costs can be decreased dramatically). In particular, there is an attack with parallelism p that reduces the time to find an algorithm-bound PoW solution by (roughly) p. The (potentially) saving factor is that the required memory bandwidth also grows by a factor of p. Thus, this particular attack may be unrealistic in practice (assuming that no ASIC/memory chip can be developed that accommodating). Proposition 7 does leave open the possibility of other (yet unknown) practical parallel attacks.

- We have added a totally new discussion on parallel attacks in Section VI.B. However, the practicality of them seems to be pure speculation.

Minor:

Footnote 1: "are now integrated"

The authors only cite their own paper [15] during their discussion of time-space tradeoffs of MHFs in the introduction. See High Parallel Complexity Graphs and Memory-Hard Functions. Alwen, J. and Serbinenko, V. STOC 2015.

(More recent)

Balloon Hashing: a Provably Memory-Hard Function with a Data-Independent Access Pattern. Boneh Corrigan-Gibbs, Schechter. https://eprint.iacr.org/2016/027.pdf

Efficiently Computing Data Independent Memory Hard Functions. Alwen, J. and Blocki, J. CRYPTO 2016.

- added

Reviewer B:

On the whole I think this submission merits publication at this venue. It is relatively well written and makes a nice contribution to the area of ASIC resistant computation which is itself undoubtedly a well motivated topic of research in the field of cryptocurrencies.

I particularly like the idea of using the generalized birthday bound as a basses for memory-hard problems. From a theoretical perspective the problem has been previously studied in various forms which provides us with various tools (algorithms) and related lower-bounds for reasoning about the construction. More practically it is simple to describe, understand and implement and the resulting asymmetry between prover & verifier are extreme. It also seems to lend itself well to tuning various lower-bounds on the resources required to produce proofs. It is also very helpful that that an implementation is available.

However I do have two concerns with the current writeup and I **strongly** suggest they should be addressed before the final version is submitted for publication.

1) The writeup is quite unclear about what is being conjectured vs. what is being proven when it comes to security. It would benefit greatly from a paragraph summarizing what is know, what is shown and what remains open/is being conjectured.
In particular my understanding is is that in order for the construction to be a good PoW according to the authors own criterion it must be that no algorithm solves the "algorithm-bound" version of the k-XOR problem significantly better then the improved Wagner's algorithm and that no better optimizations exists for Wagner's algorithm then the ones listen in the writeup. Only then do we get the guarantee that the construction is "Optimization-free". In

other words, only then do get the guarantee that an adversary has no advantage over the honest parties. In my opinion this is an absolutely crucial feature of any PoW construction. Why this should be true for the proposed construction should be made much clearer given it's importance.
Indeed the closest I could find for any such justification may be the sentence "Therefore, if we pre-fix the positions where 2^l-XORs have zero bits, we bind the user to a particular algorithm flow." But this sentence makes little sense to me (which is why I think it needs to be either much expanded upon, or else out right removed). On a high level, my understanding of the authors intentions here are based on the observation that Wagner's algorithm produces a particular subset of possible solutions to the k-XOR problem so the authors modify the problem to only accept solutions from this subset. However to then make the leap that this means **any** adversary must necessarily use Wagner's, or even just a "Wagner-like" algorithm is a very big leap. In particular I think that A) "algorithm flow" or "wagner like" or similar terms should be made much more clear and precise and B) this conjecture upon which security is based should be made very explicit and highlighted. After all it is an important starting point for future work wishing to confirm or refute the security of the protocol proposed in this work.

To be clear, I am by no means asserting there to be a problem with the conjecture, or even that there is somehow good reason to believe it is false. Simply that if such a conjecture is needed for full security then it should be stated both explicitly and precisely. (Though I suspect making it precise may turn out to be not be that easy... *e.g.* is it clear that no third type of optimization, other than the two listed at the top of page7, can not exist for Wagner's algorithm? Put differently how can one characterize the set of algorithms for which security is actually shown in the submission?) Any evidence why such a conjecture should hold would also be greatly appreciated and significantly strengthen this work.

- We have discussed this in a new subsection V.D

2) The introduction repeatedly confuses memory-bound and memory-hard functions. These are not the same and using the terms and corresponding results interchangeably should definitely be avoided. Memory-bound means many cache misses are needed to compute the function. "Memory-hard" means a lot of memory is needed to compute the function. While the later may (at least in some sense) imply the former the converse is not true. For example I strongly disagree with the discussion concerning [22]. [22] does the same (*i.e.* uses the same model, security definition and a closely related proof) as [20] and has the same intuitive goal and motivation as [6,20]. Yet in the submission, [22] is described as being "memory-hard" while [6, 22] are said to be "memory-intensive" (as an aside: why introduce a new term here? why not use the existing one used in those and other works: "memory-bound"?). However even a brief inspection of [22] and [20] show that they are actually _very_ similar results. Indeed [22] is formulated explicitly as improving on the construction of [20]. More generally to see the difference between memory-bound and memory-hard consider a machine with say 1MB cache, doing random pointer jumping in a 2MB array of incompressible bits (*i.e.* essentially the construction of [20] and [22]). This is a memory-bound function as computing it (with any algorithm) results in many cache misses (as proved in [22, 20]). However it is not memory-hard in that it doesn't require a lot of memory (only 2MB). To be clear: the goal of

[22] (and [6, 22]) was **not** that an adversary requires a lot of memory. just that they require a lot of memory accesses. But this can already be achieved with not very much more memory than cache. Indeed, this is why increasing the difficulty parameter for such memory-bound function constructions only increases the number of pointer jumps, but not the size of the array in memory.

- We have clarified the introduction on these points.

Other small comments

- Abstract: "The proof-of-work" -> "Proofs-of-work"

- Page 1: Botnets seem a bad motivation for this since they consist of the same type of hardware as honest users. given that botnets are used for other intensive computation (such as bitcoin mining) it would seem that hiding intensive resource usage from an infected hosts owner is not too great a problem (especially given the extreme under-usage of the hardware in most home PCs.)... ASIC/FPGA resistance is a far better motivation for this submission.

- added

- Page 2: "is expected" -> "we expect" Also [23] has been implemented and is available as part of the github project working towards a full spacemint implementation. If the submission wants claim that [23] is not practical it needs to explain what is wrong with the current implementation.

- added

- Page 2: It would be good to explain why not being amortization-free should be considered a problem for the Proof-of-Space protocols in [23] given that [37] uses them in precisely one of the main applications motivating the PoW in the submission: namely a cryptocurrency. Also I don't understand what is meant by "the computational penalties are guaranteed only after the memory reduction by a logarithmic factor". This should be made clearer and it should also be clarified why the authors believe their results address this problem. (If the statement refers to what the authors in [23] were able to prove in terms of a security statement then I feel this to be rather misleading criticism of [23]. After all 1) no "attack" on [37] is known to exist between the gap of what is proven and what one would ideally desire and 2) I am unclear on why the submission should have a significantly better state of affairs in terms of actual security guarantees. As discussed above, my understanding is that security of the submission is based on a strong lowerbound conjecture (not unlike [23] which also relies on a conjecture). Given the reliance on, a priori incomparable, conjectures criticising exact security seems... a bit misleading.)

- added

- Page 3: in the numerator in the equation "M/q" -> "M_S0/q"

- The authors highlight a useful property called "Progress-free Process" but then never refer to it again. It might be nice to add in a comment somewhere high lighting why the submission enjoys this property (or at least that it does).

- The progress-free property of Equihash highly depends on the parameters, and we believe that for those taking less than a second it can be considered progress-free, and for others it is discussible.

- I'm slightly unclear as to why the authors believe that a bitcoin like PoW but using say Argon2d (being faster then scrypt to avoid the Litecoin problem) would not constitute a PoW according to their criterion. For example such a protocol exhibits the required asymmetry: The verifier only performs a single evaluation of the hash function compared to the exponentially many evaluations by the prover. In fact such a protocol would also be progress-free, allow for tuning various resource constraints and lend itself well to implementation. I doubt I will be the only one with such thoughts so I think it would help if the authors address them.

- added

ISSN 2379-5980 (online)
associated article DOI
10.5915/LEDGER.2017.48