BeamHash II Specification

Wilke Trei

June 17, 2019

Contents

Introduction

Proof-of-work (PoW) is a common central concept to secure cryptocurrencies as well as blockchains in general. In this document we introduce a new family of PoW schemes that are based on the Equihash proof of work scheme. One member of this family named *BeamHash II* will be used as new PoW scheme for the cryptocurrency Beam [1] from the planned fork in July 2019. Due a new set of parameters for the *EquihashR* family of PoW schemes it also allows to classify the previous proof of work *BeamHash I* as a member of the new family.

Equihash is a family of PoW schemes that were introduced by Alex Biryukov and Dmitry Khovratovich in 2017 [3]. Equihash is a computationally asymmetric proof of work scheme that offers a fast verification of a computationally hard problem. The schemes computational complexity as well as its memory footprint can widely be parameterized.

In this document we will first provide the basic definitions around Equihash and analyze the impact of the Equihash parametrization on the memory consumption as well as the computational complexity. Also we will discuss strategies on special purpose hardware implementations and collect arguments which ranges of parametrization may be feasible for hardware implementations with modern technology.

In the second section we will review the change made to stock Equihash 150/5 towards *BeamHash I* and introduce the new parametrization set that gives raise to the *EquihashR* family of PoW schemes and to *BeamHash II* in concrete.

1 Equihash

1.1 Definition of an Equihash Solution

Despite its name Equihash is no hash function in the narrow sense. Instead an Equihash solution is the solution to a mathematical problem. The problem can be formulated like follows.

Definition. *Let n and k be the Equihash parameters and let work and nonce be given bit streams. Then we define $m := \frac{n}{k+1}$ to be the* collision length *for n and k. Further let*

$$B(k) := Blake2b\left(\text{concat}\left(work, nonce, l\right)\right)$$

be the output of the Blake2b hash function for $l = 0 \ldots \frac{2^{m+1}}{\lfloor \frac{512}{n} \rfloor}$.

Out of each $B(l)$ one computes $s := \lfloor \frac{512}{n} \rfloor$ disjoint sections of length n bits and index them first in order of l and then in their local position within $B(l)$.

Then a valid solution of the Equihash problem is a set of 2^k indexes such that

(a) all indexes are pairwise distinct.

(b) for any $1 \leq i < k$ the exclusive or (xor) of all elements referenced by an 2^{i+1} elements index block is 0 on the first $i \cdot m$ bits.

(c) the exclusive or of all elements indexed by the solution equals 0.

(d) the indexes are sorted in a way such that for two index blocks with 2^i indexes each the one with the lowest leading element leads first. E.g. $I_{2 \cdot j} < I_{2 \cdot j+1}$ for all $0 \leq j \leq 2^{k-1}$.

For the sake of completeness we must mention that when computing the s sections the starting points of each segment are byte aligned. Since 512 is a power of two and one byte equals 8 bits this byte alignment has no impact on the number s, but changes the position and value of some of the segments in case $8 \nmid n$.

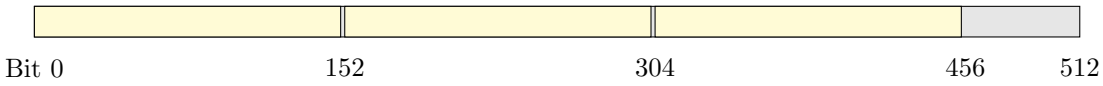

Figure 1: Positioning of Equihash 150/5 segments in a single 512 bit Blake2b output with padding bits

We immediately note the asymmetry between the computation effort of a solution and its verification. To verify a given index set we only require to check the conditions on index ordering and compute the 2^k segments of Blake2b output and xor them in the right order to validate the index set is a valid solution. For the generation of such a solution we must consider all 2^{m+1} Blake2b output segments that are allowed. In the next section we will line up the most common algorithm for generating valid index sets.

1.2 Equihash Mining Algorithm

The central concept of Equihash verification and mining are so called *step rows*. A step row is composed of the n bit wide segment mentioned in the previous definition, that is segmented into $k-1$ segments of size m and one final of size $2m$. Alongside, the index of the segment is stored. The pattern of the segmentation of the n bits is illustrated in Figure 2.

The algorithm behind Equihash mining is the Wagner algorithm [6]. Its core functionality is given by combining step rows in overall k rounds. This is done by sorting all step rows by their most significant remaining sub-segment of either m or $2m$ bits. If for two step rows in round i their corresponding sub-segments are equal, an output step row is created. This output has only

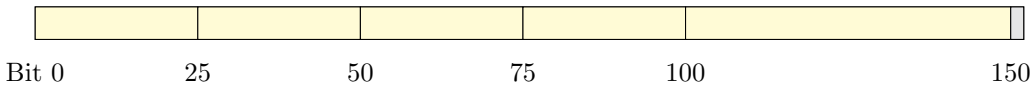

Figure 2: StepRow pattern for Equihash 150/5

$k-(i-1)$ segments left, which are the binary xor of the inputs sub-segments. Furthermore the newly created step row got an index list that is the merging of the two input index list. Hereby the order of indexes of the input lists are preserved. The list of the 2nd input will be places after the list of the first input if and only if the lowest most index in the first list is lower then in the 2nd. Note that by iterative application it is always guarantied that the lowest index that gives a step row is placed in the first position of its index list.

Figure 3 illustrates the combination of two step rows in round 2.

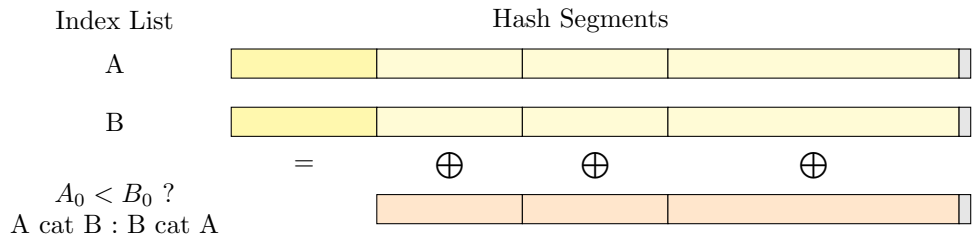

Figure 3: StepRow combination in round 2 of Equihash 150/5.

The following theorem can be used to estimate the number of outputs of one round giving the number of input segments.

Theorem. *Let $s_1, \ldots, s_l$ be randomly chosen set of bit string of length z.*
Then the expected number of pairs $\#\{(i,j) \mid i<j \text{ and } s_i = s_j\}$ is about $\frac{l^2}{2\cdot 2^z}$.

Proof. The total number of pairs $(i,j) \;||\; i<j$ is about $\frac{l\cdot(l-1)}{2}$. We note that $s_i = s_i \leftrightarrow s_i \oplus s_j = 0$. Since $s_i \oplus s_j$ randomly takes one of 2^z values about $\frac{\frac{l\cdot(l-1)}{2}}{2^z} \approx \frac{l^2}{2\cdot 2^z}$ of the pairs satisfy the condition. □

In the concrete situation we have 2^{m+1} step rows generated from Blake2b input. These are matched on m bits, so the first sub-segment of each hash row is a member of a set of size 2^m. Therefore in the first round the output is expected to be

$$\frac{\left(2^{m+1}\right)^2}{2\cdot 2^m} = 2^{2m+2-m-1} = 2^{m+1}$$

step rows with remaining $n-m$ bits. We see that m was chosen in a way that the expected number of round 1 outputs equals the number of input step rows.

Iteratively we can conclude that we also expect to have 2^{m+1} step rows after the $k-1$ th round. In the last round we then need to match on $2m$ instead of m bits. Therefore the expected number of outputs of the last round is

$$\frac{\left(2^{m+1}\right)^2}{2\cdot 2^{2m}} = 2^{2m+2-2m-1} = 2.$$

Here we see an substantial difference between a classical hashing algorithm and the Equihash PoW puzzle. A classical hashing algorithm would produce a fixed number of results in each

iteration, while for Equihash we only have the expected number of two solutions per iteration, which is thus not guarantied. It may as well be that an Equihash instance has no solution as that the miner is lucky and finds more then expected.

A central aspect of the estimates in the theorem is the randomness in the pair creation. This is jeopardized in case that one step row is created twice over different pathes, because then one will gain a completely zero step row in the next round. The following example illustrates how such an element can be created in at least 3 rounds. We see that the condition that gives raise to a full zero step row are in fact common enough to jeopardize the equal distributed randomness assumption of the theorem.

Example. *Consider the situation where at least four step rows with index sets A,B,C,D will be bucketed into the same bucket, i.e. their next m bits wide segment equals. Without restricting generality let this be in round 1.*

Then the round code will create all combinations of this step rows, which have index sets equal to AB, AC, AD, BC, BD and CD without consider the ordering.

If now any two of this 6 elements match on the next m bits as well we run in danger of degenerated solutions. Let AB and CD be this two elements, then we know that the xor of the round 1 and round 2 matchbit is zero given this indexes. But because of the elements coming from the same round 1 iteration we can conclude that AC and BD as well as AD and BC also will be combined in round 2 all giving raise to ABCD in the output of round 2. So in round 3 we will create multiple elements where all further match bits are completely zero.

The condition is not so rare that it can be ignored. Therefore from round 3 onward all miners implement a check if an output step row is already completely zero. This also rules out the cases where e.g. AB and AC will be matched. It can be observed that the test if a step row already equals zero - which almost exclusively happens when it is composed of duplicate elements in its index tree - is more and more important for higher round numbers. This is likely due to the number of index tree elements doubling with every round.

1.3 Resource Consumption of Equihash Mining

As can be seen in the previous section the main influence on the memory consumption of the algorithm is the number of step rows that act as in and output of the matching rounds. We must note that the index tree appears to be of increasing size by doubling the number of entries in each round, but a common optimization is to store it separately from the matching hashes and only store references to the previous tree layers in later rounds. This way even in later rounds for each step row only the indexes of its two ancestors get stored.

As a consequence we can estimate that the memory consumption is in the complexity class $\mathcal{O}\left((k+n)\cdot 2^{m+1}\right)$. The following table lists the memory consumption of the Equihash implementations that were used as crypto-currency PoW.

n	96	200	210	144	192	125	150
k	5	9	9	5	7	4	5
Coin [a]	MinexCoin	ZCash	AION	BitcoinZ	Zero	ZelCash	BEAM
m	16	20	21	24	24	25	25
Step Rows	2^{17}	2^{21}	2^{22}	2^{25}	2^{25}	2^{26}	2^{26}
Memory Use [b]	≈ 7	≈ 200	≈ 600	≈ 1500	≈ 2800	≈ 3000	≈ 3200

Table 1: Examples for Equihash instances

[a] First appearance of the Equihash parameters as primary PoW
[b] In MBytes per instance for current GPU efficient implementations

The time consumption of Equihash mining naturally depends on the number of step rows as well as the number of rounds. The exact timings of each round may in practice vary between

mining software, hardware vendors and last but not least from round to round, because the step rows decrease in size in later rounds. For the Equihash instances with 3 step rows per Blake2b output, e.g. 144/5 and 150/5 we observed that the duration of the Blake2b phase of the algorithm approximately equals the duration of the first two rounds by using the mining software lolMiner. As a consequence the timing of the Blake2b phase is approximately 20% of the total execution time of this algorithms with total 5 rounds. This is because the last round requires to write much less elements compared to any previous round and thus is less time consuming.

With respect to energy consumption the rounds are less balanced. The matching rounds are dominated by xor operations and a bucket sort to prepare the input of the next round. This is computationally very lightweight and overall dominated by the duration of fetching and writing the elements to be combined. On the other hand the Blake2b filling of the step rows is very computational intensive. As a consequence during the execution the differences between average and peak consumption is rather high.

1.4 Equihash on Special Purpose Hardware

Current ASIC implementations of the Equihash 200/9 algorithm make use of large amounts of on chip memory offering a low latency and high memory bandwidth. For example the Bitmain Z9 Mini [2] features 144 Mbyte of on Chip memory which is close to the theoretic minimal memory for performing the algorithm on a CPU. Given the performance of the ASIC the used bandwidth exceeds 5 TBytes per second.

In contrast an efficient GPU implementation will usually use a larger memory footprint because the off chip memory controllers benefit from read and write access that is at least 32bit, better 128 bit aligned. Off chip memory has a bandwidth ranging of about 256 Gbyte/s on medium range GPUs to trice the value on high end GPUs equipped with HBM2 memory.

Beside the advantages of high amount of on chip memory and better packed access patters an ASIC as well as an FPGA can trade time spend for parallelizeable compute operations by chip space and power consumption by assigning more circuits to the task. For Equihash the fraction offering most potential for this trade is the Blake2b algorithm.

A massively increased performance of the Blake2b algorithm may be beneficial to trade compute power against bandwidth like follows. In the first rounds of the Equihash matching phases less bits then required for performing the full algorithm can be stored and loaded. As soon as the abandoned bits get required for continuing with the algorithm the chip can recover them by computing the hashes again from the indexes carried to this round.

This approach is in particular efficient in the early rounds when the bandwidth required to transfer the indexes is low compared with the transfer of the original workload bits. Also in this early rounds less different Blake2b passes have to be used to recover the concrete bits. For GPUs this approach is not an option, because adding Blake2b computations increases duration instead of space of the algorithm. Also this operations consume too much time to be hidden when performing memory operations and they would increase power consumption as well.

1.5 Applicability towards Equihash 150/5

In the previous section we defined three potential benefits of ASICs compared to GPUs that are large on chip memory areas, a better packing and coalescing of off-chip memory operations and time-space algorithm trade-off.

Regarding the first issue we have strong confidence that a specialized chip for Equihash 150/5 will not take this exact approach due to its increased memory footprint. When completely computed the output of the first round is $(26 + 150) \cdot 2^{26}$ bits and thus approx 1.4 Gbytes. We claim that with growing index tables of matched elements this is a sharp lower bound. This even

applies when using the trick described in the time memory trade-off part of the previous section, because even the indexes written in the first 4 rounds compressed to 36 bits per matched element plus the required remaining bits for performing the final matching round requires at least this space.

In practice we even can estimate that the real memory consumption of an efficient implementation is at least twice the size, similar to the ZCash parameters.

Therefore a single chip ASIC holding enough memory to perform all operations at Tbytes per second bandwidth range would be of 10 to 20x the size of the chip to perform ZCashs Equihash 200/9 algorithm on the same manufacturing process. This would increase the cost for producing the chip dramatically, because on the same production processes a higher yield can be expected and also larger chips are more costly.

Of cause this approach may and will get feasible in the future with advanced production methods and new technologies regarding fast on chip memory or staking chips with height bandwidth. Never the less the mentioned amount of memory is high enough that the time to market of such new inventions will be longer then our planned PoW review and adjustment period.

As of summer 2019, the PoW mining ASICs with the largest on chip memory are the Obelisk GRN1 [5] and the Innosilicon G32-500 [4], which are both designed for mining Cuckatoo-31+ and are supposed to be available in 4th quarter 2019. Although the specs are not fully public it is known these devices offer 512 MByte on chip scratch memory at a 16 nm structure and belong to the largest chips that can be created at this structure process. We conclude that even with the smaller 7nm manufacturing process the approximately 2GByte lower bound of an efficient Equihash 125/4 implementation are out of reach.

The second benefit is of architectural nature. ASICs as well as FPGAs are able to perform memory operations that use bit length that are not a multiple of 32 more efficiently. Also adding more circuits in the implementation to ensure a better memory coalescing when using external memory chips is exclusive to these chips, while a GPU can improve its memory access patterns only by using software solutions.

We therefore believe this benefit can not be mitigated directly by an algorithm change unless changing the parameters in a way that most memory operations are forced to be in ideal range for GPUs. This is out of scope for Equihash with nowadays capacities.

Anyways we claim that the total effect of this benefit is limited by the capacities of the external memory controller. For the already well tuned Equihash 144/5 parameters it is known that a single run requires approximately 5 Gbytes of memory bandwidth in total. Each run will produce 2 solutions on average. Modern implementations can run up to 60 solutions per second on an unmodified Nvidia GTX 1080. Therefore on this cards about 150 Gbytes/s of the total available 352 Gbyte/s are effectively used. So a specialized chip equipped with the same memory but more efficient memory access patterns may have a gain limited to a factor of about 2.5 if the algorithm behind is forced to write the same data.

This enforcement aligns with the potential space to memory and bandwidth trade-off discussed before. The BEAM project proposes an algorithm change that makes the trade-off more costly in terms of space requirement and power consumption of the resulting chip.

Note that specialized devices that are programmable as GPUs but are reduced to and specialized for the required operations to mine crypt-currencies and offer additional optimizations for memory access still may be able to perform the algorithm 2-3x faster then a GPU equipped with the same memory and that at a potentially lower energy consumption. On the long term preventing such devices from mining the algorithm is not feasible. At the time of writing no such designs are publicly available nor is there a concrete announcement of such, so we assume that the desired head start for GPU mining BEAM is given.

2 BeamHash II & EquihashR Family of PoW

2.1 BeamHash I Data Path Change

By the nature of the Equihash 150/5 algorithm it has a blocking rate of 3 in its Blake2b phase. This is that if the original algorithm is modified in a way that in later rounds the Blake2b algorithm has to be performed again, for each step row we require 3x the computation amount of the initial calculation, because in the first round 3 hash strings of 150 bit length are generated in one computation. This also applies to the verification of an Equihash 150/5 solution, but is no issue for the verification, because the total number of Blake2b runs to verify one solution is 32 while the average number of runs for generating one solution exceeds $11184810 \approx \frac{1}{2}\frac{2^{26}}{3}$.

In order to cover the mentioned time memory trade of given by allowing more computations we propose to increase this blocking factor. This will improve the original scope of Equihash to achieve an algorithm binding and so increase the resistance against diverging approaches.

Algorithm. *Let $B(l)$ be the 512 bit string corresponding to the Blake2b hash for input index l and let $B(l)_j$ be the j-th 32 bit component interpreted as 32 bit integer. Then let $B'(l)$ be the 512 bit string computed like follows:*

$c \leftarrow 16 \cdot \lfloor \frac{l}{16} \rfloor$
$B'(l) \leftarrow 0$
while $c \leq l$ ***do***
    $B'(l)_j \leftarrow B'(l)_j + B(c)_j$ *for all* $0 \leq j < 16$
    $c \leftarrow c + 1$
end while

For Beam Hash I - which is based on Equihash 150/5 this data-path change increases the blocking factor from 3 to a value between 3 and 45 depending on the exact index of the step row. In order to compute the modified hash string B' it may be required to know all 15 other hashes in the same group of 16 hash elements. On a GPU this can be cheaply achieved by using the local memory on the device in the initial computation phase. Also on modern GPUs neighbored threads with index only varying on the lowest four bits run in lock steps, so the sharing over the local memory does not introduce new synchronization barriers.

Therefore for performing the algorithm the intended way the slow down by the extra computation of the sums will be negligible. On the other hand when it is intended to run Blake2b later again to recover bits from known indexes this is up to 16 times more costly then before and overall up to 48 times so costly then in the initial round with an average extra cost factor of 24.

By this approach we aim towards forcing the algorithm to follow the same algorithm implementation as done on GPUs or else to use more chip-space and drastically increased power consumption, because performing the Blake2b algorithm is the most power consuming component of Equihash.

Note that also the verification of solutions get more costly by an average factor of eight. But since only few solutions needs to be verified and due to the still high asymmetry of generation effort compared to verification effort the drawback is acceptable. Overall with this modification the verification of an Equihash 150/5 solution can be done at lower average cost as the verification of an Equihash 200/9 solution while the worst case costs are equal.

2.2 EquihashR Family Definition

We introduce an additional parameter r to the Equihash prove of work scheme that changes the behavior of the hash generation and the step row matching. This parameter is used to define the new EquihashR family of PoW that should have reduced computation costs while offer a higher algorithm binding and thus resistance against diverging approaches.

Definition. *Given integer parameters n,k and r.*

Then we define the EquihashR<n,k,r> proof of work scheme to equal Equihash<n,k> with the following modifications

(a) The change in the hash calculation described in section 2.1 is applied to the hash calculation.

(b) Instead of 2^{m+1} step rows we create 2^{m+1-r} ones in the Blake2b phase of the algorithm. The generated step rows equal those in the original algorithm, but we restrict to the first indexes until the upper cap is reached.

(c) The first $2 \cdot r$ bits of each n bit step row are reset to 0.

The rest of the algorithm is kept as for the original Equihash, so as a conclusion to (c) the first round only matches $m - 2r$ bits.

The parameter r reduces the amount of computation required to fill the array of hash values, because only $\frac{1}{2^r}$ of the original step rows are calculated. Therefore the compute intensive component of the algorithm is reduced significantly.

In round 1 the rather low number of hashes is matched on only $N - 2r$ bits. As a consequence following the theorem in section 1.2 the expected number of outputs of round 1 is

$$\frac{2^{2(m+1-r)}}{2 \cdot 2^{m-2r}} = 2^{m+1}.$$

Therefore the expected number of output of round 1 is the same as for the original Equihash algorithm. Also the memory requirement of the algorithm is unchanged, except for a lower size index tree of round 0. Our scope for introducing r is to reduce the dependency of the PoW scheme on the computational power for the Blake2b generation. The idea is that existing GPU miners profit from a lower power use while one of the domains where specialized hardware has an edge over GPUs - computational heavy tasks - gets less impact on the solution calculation.

As a drawback to the original algorithm for each index the number of round 1 step rows incorporating the index increases from 2 to $2 \cdot 2^r$ on average. Therefore the parameter r should be used moderately in order not to jeopardize the randomness assumption of the theorem in section 1.2.

2.3 BeamHash II

For the scheduled Beam hard fork we picked the EquihashR parameters $n = 150$, $k = 5$ and $r = 3$, i.e. a factor of 8 reduction in the Blake2b component. Figure 4 illustrates the modified step row pattern for BeamHash II. Here the shortened step row that will be matched in round 1 is colored in orange.

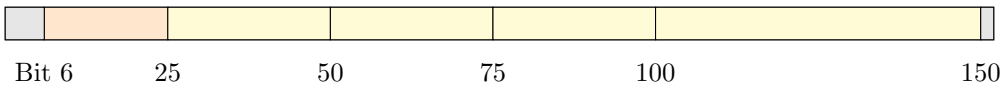

Figure 4: StepRow pattern for r=3 (BeamHashII)

All other aspects of the algorithm will remain unchanged. Especially this is valid for the size of the solution size and thus the stratum protocol of the Beam currency. Note that in theory the number of bits of a solution would reduce from $32 \cdot 26$ bits to $32 \cdot 23$ bits. We consciously discarded the reduction for the implementation in the Beam cryptocurrency in order to ease the implementation overhead for miner software developers and pools and to reduce a potential error source.

2.4 Empirical Analysis

In order to evaluate the parameter r we started a series of computations based on BeamHash I, i.e. EquihashR 150/5/0, where we tested a wider range of values for r. For each tested value of r we checked at least $10'000$ nonces to gain a decent sample size. Note that we did not generate exactly 2^{26-r} step rows. Instead the number was slightly lowered in order to make the number of Blake2b calculations dividable by 256. This had some advantages in our GPU implementation.

The following table gives the average number of step rows left after each round as well as the standard deviation and the average number of solutions per iteration for different values of r.

		Blake Round	Round 1	Round 2	Round 3	Round 4	Solutions
BeamHash I	Average	67'108'608	67'108'440.8	67'108'161.7	67'107'431.0	67'106'132.9	1.989
	Std. Deviation	-	8'080.06	18'101.75	36'703.02	73'731.33	-
r=1	Average	33'553920	67'108'892.9	67105068.0	67101323.7	67093913.1	1.971
	Std. Deviation	-	8'232.58	18'420.57	37'846.57	76'058.62	-
r=2	Average	16'776'960	67'106'527.9	67'104'287.7	67'099'851.3	67'090'942.7	2.026
	Std. Deviation	-	8'239.37	18'586.03	38'114.90	76'684.49	-
BeamHash II (r=3)	Average	8'388'096	67'098'069.8	67'087'248.5	67'065'681.5	67'022'639.6	2.015
	Std. Deviation	-	8'195.76	18'288.52	37'405.58	75'143.71	-
r=8	Average	512'776	66'977'746.5	66'846'936.1	66'584'134.8	66'063'507.7	1.924
	Std. Deviation	-	8'072.35	17'916.2	36'524.42	73'180.23	-

Table 2: Average number of generated step rows for EquihashR 150/5/r and selected values of r, 10'000 simulations each

We observe that for the selected range of r the statistical properties are very stable. For higher values we observe that the mentioned filter that tries to detect step rows with duplicate indexes is more active. Never the less from $r = 9$ and higher we observed that the number of candidates that pass round 5 but then get discarded due to duplicate indexes is exponentially increasing. At the time writing it is not yet clarified if this effect is due to a lack of generality in our implementation or due to the effects of the index tree, which has a highly increased number of step rows including a fixed index after round 1. Deeper investigations will be required to cover these cases and analyze the behavior of EquihashR for high r.

Regarding the energy consumption we did a test with a Nvidia GTX 1080 founders edition and an AMD Radeon R7. Both GPUs were tested with their stock clocks and the OpenCL reference miner. Note that this is miner is generally a bit better optimized for AMD devices. The following table gives the ASIC power consumption and hash rate for both algorithms supported by the miner.

	BeamHash		BeamHash II	
	Perf	Watts	Perf	Watts
AMD Radeon 7	17.5 sol/s	206 W	23.6 sol/s	175 W
Nvidia GTX 1080	8.5 sol/s	132 W	11.3 sol/s	118 W

Table 3: Performance and power consumption of the reference implementation on selected hardware.

We observe that the new parameter set yields a lower energy use by 10 to 20% although the miner is able to produce about 30% more solutions per second. Also this behavior is consistent between the two GPU vendors. We thus have strong evidence that BeamHash II matches our design scope.

References

[1] BEAM — Mimblewimble-based Privacy Coin, https://beam.mw

[2] Bitmain Antminer Z9 Mini, bitmain.com

[3] Biryukov, Alex; Khovratovich, Dmitry (2017). "Equihash: Asymmetric Proof-of-Work Based on the Generalized Birthday Problem: Open Review", https://ledger.pitt.edu/ojs/index.php/ledger/article/view/48

[4] INNOSILICON G32, https://www.innosilicon.com/html/grin-miner

[5] OBELISK GRN1, https://obelisk.tech/products/grn1.html

[6] Wagner, David A Generalized Birthday Problem. Lecture Notes in Computer Science 2442 (2002) 288303