Annales UMCS Informatica AI XII, 3 (2012) 25–35
DOI: 10.2478/v10065-012-0024-0

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

Attacks on StreamHash 2

Mateusz Buczek[1]*

[1] *Institute of Mathematics and Cryptology, Faculty of Cybernetics, Military University of Technology*
Kaliskiego 2, 00-908 Warsaw, Poland

Abstract – StreamHash 2 is a hash function proposed by Michał Trojnara at the Cryptography and Security Systems in 2011 Conference. This algorithm is a member of StreamHash family which was first introduced in 2008 during the SHA-3 Competition. In this paper we will show collision attacks on the internal state of the StreamHash 2 hash function with complexity about 2^{8n} for the $32n$-bit version of the algorithm and its reduced version with complexity 2^{8n}. We will also show its application to attacking the full StreamHash 2 function (finding a collision on all output bits) with complexity about 2^{88} . We will try to show that any changes made to the construction (for instance the ones proposed for StreamHash 3) will have no effect on the security of the family due to critical fault build into the compression function.

1 StreamHash and the SHA-3 Competition

On November 2nd, 2007 the National Institute of Standards and Technology (NIST) announced a hash function competition for a new SHA-3 (Secure Hash Algorithm). The goal of the competition was to replace the older constructions such as SHA-1 and SHA-2 in all their variants with a new, more secure and faster algorithm. Another goal of the competition was to improve knowledge in the field of hash functions and find new attacks and new constructions for hash functions.

There were over 50 proposed algorithms and 51 of them were selected for the first round. One of those 51 candidates was StreamHash (now, due to the family development, called StreamHash 1) proposed by Michał Trojnara from Warsaw University of Technology.

*mbuczek@wat.edu.pl

Unfortunately, before the First SHA-3 Candidate Conference which was held in Leuven in late February, 2009 the algorithm was broken. Due to that it was officially retracted from the competition.

But in the past years the algorithm was improved, presented at a few conferences and in 2011 the new version of StreamHash was discussed at the Cryptography and Security Systems Conference.

2 StreamHash 1 construction

The whole family of StreamHash function is based on basically the same construction with just minor tweaks. We will start the presentation with the most basic 128-bit StreamHash 1.

2.1 Mode of operation

StreamHash is an untypical algorithm using the Markle-Damgard construction. The message block is disproportionally small in comparison to the internal sate (chaining value) of the function. Additionally, the padding function is dependent on only the last block of the message.

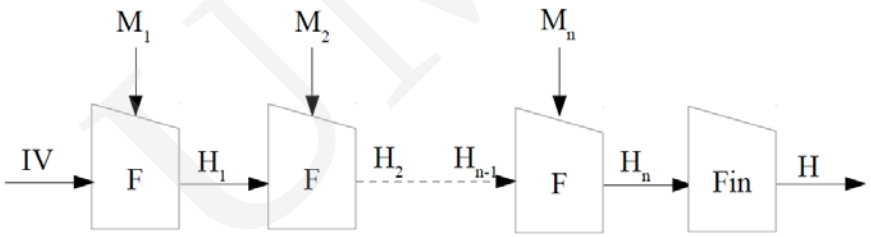

Fig. 1. StreamHash mode of operation.

In every compression function application only 8 bits of the message (blocks $M_1, M_2, \ldots, M_n$) are transformed, and the output is equal to the desired hash size ($H_1, H_2, \ldots, H_n, H$ - which should be a multiple of 32). The initial value (IV) of the algorithm is a vector of zeros of length equal to the hash size.

After the final block of the message is transformed a finalization function working only on the state H_n is applied. It consists of several applications of compression function with the message being replaced with chosen bits of the internal state. After that final mixing is applied and the produced state is the algorithm output.

2.2 Compression function construction

Every block of message is transformed by a compression function which consists of several mini-transformations (NLF) working in parallel. Every mini-transformation has 3 inputs:

- chaining value input of length 32,
- message input of length 8,
- distinguishing input of length 8.

The chaining value inputted into the compression function is divided into n 32-bit blocks and processed in parallel with the NLF transformation.

The only difference between the parallel lines of transformation is the 8-bit lane number.

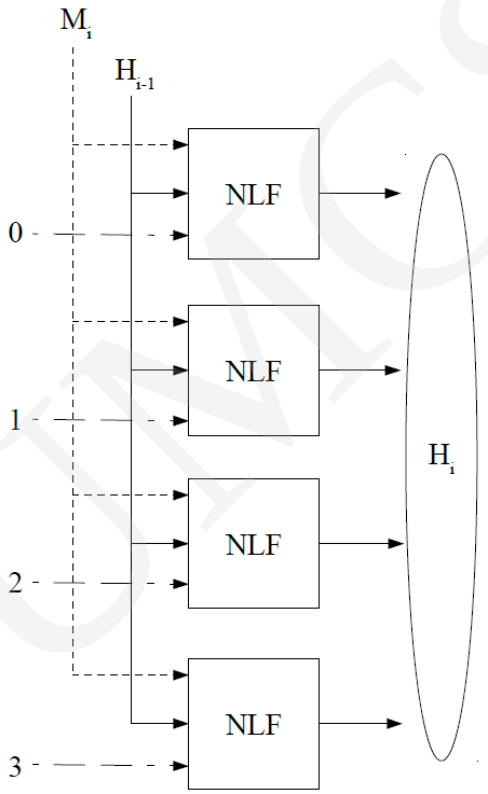

Fig. 2. StreamHash compression function for the hash length of 128 bits.

2.3 NLF transformation

NLF transformation works as follows:

- 8 least significant bits of the internal state word inputted into the transformation are xored with the lane number and the message block,
- S-box function is performed on the xor output,
- output of the xor is xored into the internal state inputted to the transformation and then outputted as a result.

S-box table used in NLF looks like this:

```
89f20430 ca107c01 88978567 32a1f12b 87eabbfe 62ab0ed7 acaa62ab c5073876
4f9274ca ff7d1382 78c1ddc9 4716ff7d 61d82dfa c01fcb59 e1e0a047 43648cf0
e52a95ad 005248d4 cd803aa2 4eb679af a41dde9c e23b49a4 01094072 bff4bac0
66d3a9b7 b72054fd 4486dc93 4568f726 7f6b0536 5e9d753f 6e4568f7 6db34bcc
95ad1834 a86f06a5 9635d9e5 2332a1f1 670aa371 dfef61d8 b4c6c731 1fcb5915
a789f204 8db4c6c7 68f72623 c7312ec3 2a95ad18 d0609096 d27f6b05 506cb89a
24a6c507 c1ddc912 7abdcd80 5a4698e2 721ee9eb b34bcc27 b89a37b2 585e9d75
107c0109 8bceec83 0aa3712c 803aa21a b679af1b b9db9f6e e4aebe5a f8e1e0a0
fb630052 4698e23b 2c42f6d6 eb3c6db3 6f06a529 138211e3 cb59152f 8acf5f84
fc55ed53 37b23ed1 0ffb6300 b0fc55ed d3a9b720 94e7b0fc 9be8c8b1 c912395b
f577026a bac01fcb 69e4aebe ddc91239 42f6d64a 06a5294c 77026a58 f37e8acf
d15170d0 0b9edfef 8191acaa 38760ffb 3aa21a43 8211e34d 312ec333 c4889785
db9f6e45 28ee99f9 e6f57702 d5b5d27f 55ed5350 1ee9eb3c 56b9db9f 3f25c2a8
b23ed151 85670aa3 7c010940 738f738f 5f844f92 6a585e9d a6c50738 198ee6f5
e34d65bc 152f4eb6 395b57da 2054fd21 9274ca10 a04716ff 0ed70df3 03d5b5d2
da7abdcd eabbfe0c 16ff7d13 3d8bceec 7e8acf5f 1cc48897 79af1b44 648cf017
de9c1cc4 d64a5ca7 d70df37e 4bcc273d a21a4364 a5294c5d 5248d419 8f738f73
5170d060 bbfe0c81 cf5f844f 1b4486dc 86dc9322 35d9e52a 70d06090 9c1cc488
aebe5a46 183428ee 53506cb8 d82dfa14 49a41dde 026a585e a1f12b0b b156b9db
41f8e1e0 f7262332 bdcd803a 9785670a 98e23b49 c2a86f06 6b053624 f6d64a5c
753f25c2 c33366d3 0c8191ac 91acaa62 fe0c8191 d9e52a95 99f969e4 2f4eb679
932294e7 149be8c8 6cb89a37 e9eb3c6d 294c5d8d 217b03d5 59152f4e 3366d3a9
ed53506c e8c8b156 3008bff4 f01787ea 11e34d65 5b57da7a f969e4ae f2043008
08bff4ba 4d65bc78 9d753f25 c6c7312e 1dde9c1c 053624a6 4c5d8db4 5d8db4c6
fa149be8 bc78c1dd 844f9274 f4bac01f 3c6db34b 57da7abd cc273d8b 0df37e8a
3ed15170 9a37b23e 7b03d5b5 2ec33366 63005248 fd217b03 712c42f6 aa62ab0e
9edfef61 60909635 12395b57 c8b156b9 af1b4486 65bc78c1 3b49a41d f12b0b9e
8341f8e1 ec8341f8 be5a4698 7d138211 ee99f969 909635d9 48d4198e dc932294
2dfa149b 0940721e 8cf01787 40721ee9 273d8bce e7b0fc55 ad183428 2b0b9edf
1a43648c 262332a1 4a5ca789 ab0ed70d 043008bf d4198ee6 a3712c42 9f6e4568
ceec8341 3428ee99 ef61d82d 0738760f 2294e7b0 a9b72054 1787eabb e0a04716
```

As we can see in every lane of compression function transformation the NLF mini-transformation is the same and the only difference between the lanes is the number used in xor. But due to the fact that the output of the xor is transformed by the S-box operation, even the slightest change in the lane number will affect all the bits of output.

It is not stated in the supporting documentation how to deal with hash lengths larger than 2kB (the lane number in the binary code will will require more than 8 bits). But hash sizes like this will not be used in the near future and our attack should be independent of the numbering used.

Below there is the NLF transformation scheme:

2.4 Padding

Padding in StreamHash consists of at most 16 bits. The last length mod 8 bits of the message are appended with zeroes to a full byte. Then another byte which contains the number of bits of the message that were used to create the previous byte is appended to the message. So if the message has 17 bits, then 14 zeroes will be added (7 to complete the byte and 7 as the start of a new byte) and then a 1 will be appended at the end (number of bits used in previous byte 1=17 mod 8).

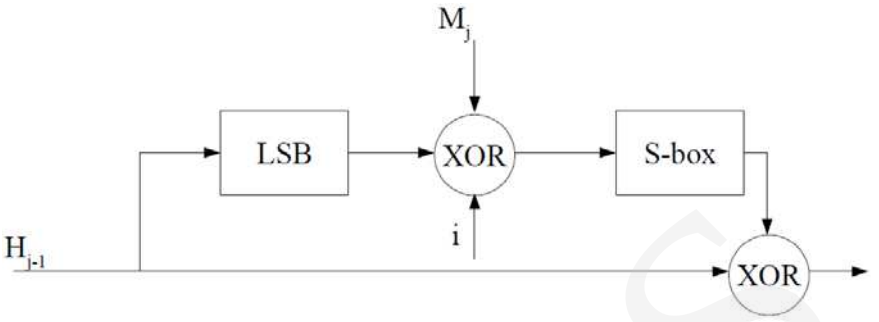

Fig. 3. StreamHash 1NLF minitransformation scheme.

2.5 Finalization function

After a final call of compression function (on the second byte of padding), a vector is built from lower 16 bits of each internal state word (in the high-endian byte order). Then this vector is used as another padding in several new calls of compression function.

Then the state words are added modulo 2^{32} to each other (the word i is added to the word $i + 1$ and the last word is added to the first one) 3 times. This is the first diffusion operation.

After that the state vector of 32-bit integers is transformed to a vector of 8-bit bytes with the high-endian byte order.

At the end diffusion operation similar to the first one is preformed. Its output is the hash function output.

3 Attacks on StreamHash 1

Several works on cryptanalysis were preformed by the community before the first SHA-3 Candidate Conference. The most significant achievement was finding a practical collision by Tor E. Bjorstad from University of Bergen, Norway. This caused the author to resign from further participation in the contest.

4 StreamHash 2

As stated before in 2011 a new version of the algorithm was presented. It includes a few changes which were supposed to strengthen the algorithm against the proposed attacks.

The biggest change is adding a Pseudo-Random Number Generator XORShift to the design. The algorithm was proposed by Marsaglia in 2003. It has a period of $2^{64} - 1$ based on a 64 bit internal state, and only 32 least significant bits of its output are taken to the function.

The output of the PRNG is added modulo 2^{32} to the previous state word and the S-box output. The adding replaces the xor operation from StreamHash 1.

This change is supposed to protect the design from falling into cycles like the previous versions, but as we will show it only increases the cycle period and the internal state collision is still quite easy to find.

Changing of the NLF transformation also affects the finalization function and this new feature is a key to StreamHash 2 security.

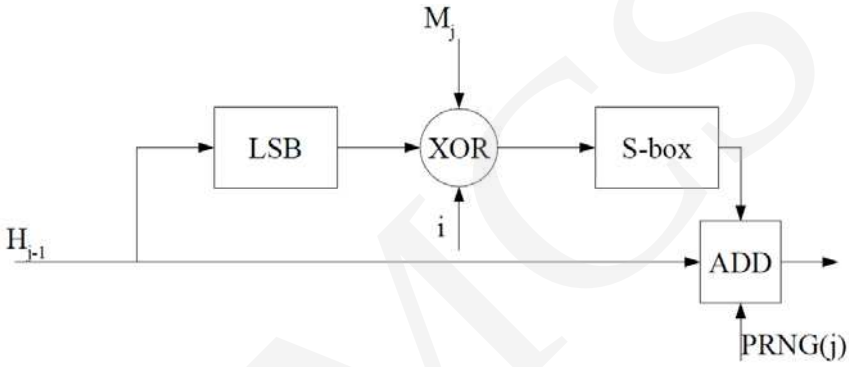

Fig. 4. StreamHash 2 NLF mini-transformation scheme.

5 Attacks on StreamHash 2

The proposed attack works in two parts. First we create an internal state of the algorithm with desired properties by injecting a few first bytes of the message (prefix). Then we iterate the algorithm with carefully chosen last bytes of message (postfix) until a collision appears.

5.1 Basic idea of attack

The basic idea of the attack is to find an internal state of the function that will not be changed in the iteration of the algorithm, or at least a part of it will stay unchanged.

Finding a fixed-point for the compression function is almost impossible due to the fact that depending on the output of PRNG we use a different function in every iteration (not completely different, but one of 2^{32} functions). Moreover, the functions repeat themselves in the same order once per the PRNGs period.

We got only 256 different message blocks and at least 2^{32} internal states so we can not even cover 1% of the outputs in the single compression function iteration.

What we will try to show is that we do not need a whole internal state collision in one iteration but a *near fixed-point*. If we are able to keep a few least significant bits of each of the internal state words unchanged, we should be able to mount a collision attack on the whole state.

5.2 Internal state requirement

Of course, the least significant bits of the state we want to keep can not be chosen randomly. We need them to fulfill a simple requirement that for every state word they xor to the same 8-bit vector with the state word number.

If this happens, the xor input of the S-box function is the same for all the parallel computational lanes. It is simply the xor of the message and the chosen 8-bit vector results from the requirement.

This will give us a full control over the S-box function and as a result, a control over the next 8 least significant bits of the state.

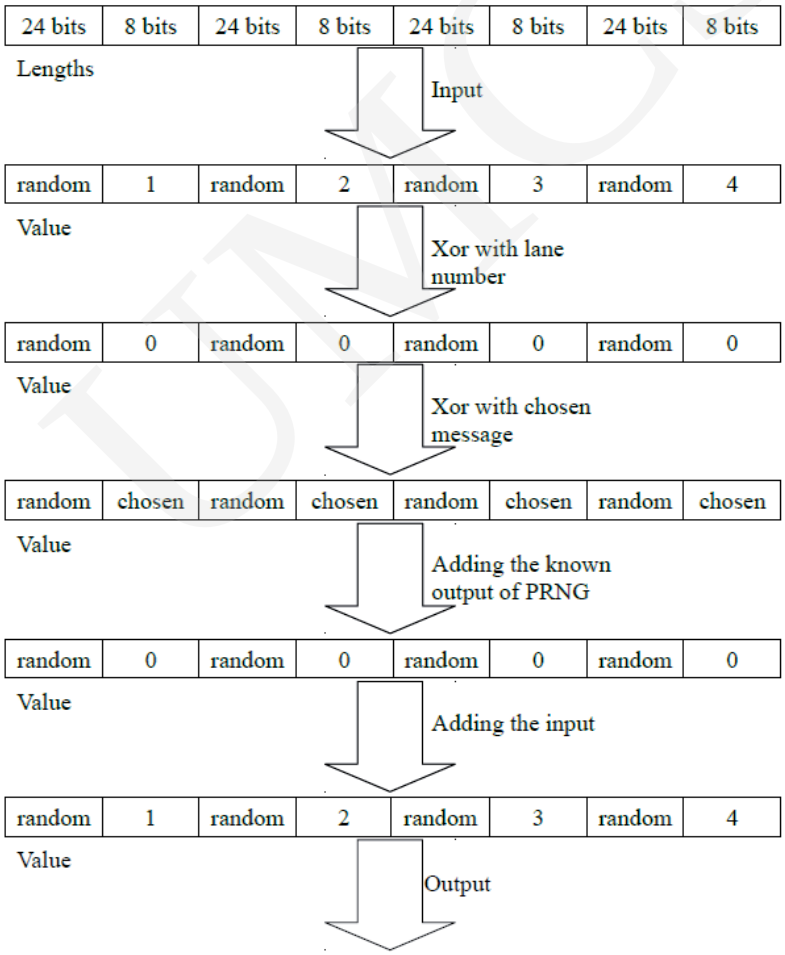

Fig. 5. Data flow in the compression function.

The compression function for every lane looks like this:

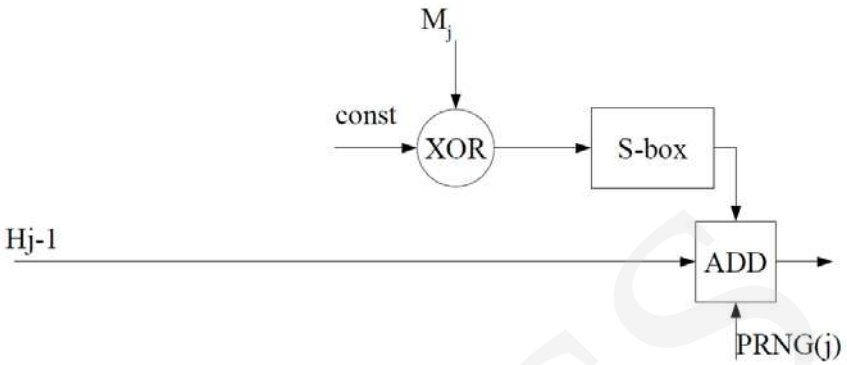

Fig. 6. StreamHash 2 NLF mini-transformation scheme while using a correct internal state for the attack.

To be honest, as we get a full control over M_j, even the xor function is non existent and can be included into S-box.

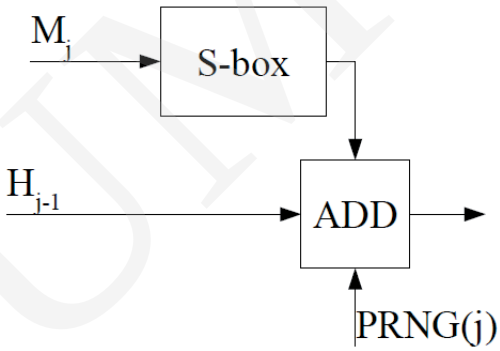

Fig. 7. Stream hash 2 NLF transformation true working scheme while using the correct internal state.

5.3 Choosing the prefixes

We do not need to choose a specific prefix. What we will do is to iterate through every message of the length 8n for 32n-bit state size which should result in finding at least one state vector fulfilling the requirements.

5.4 Iterating the function (Choosing postfixes)

As we know the goal of the attack is to keep the 8 least significant bits unchanged during the iteration. The only place those bits could be changed is the addition mod 2^{32} at the iteration end. The addition has 3 inputs:

- previous state word,

- Pseudo-Random Number Generator output,
- message block (not exactly, but as shown above the third input can be simplified to act just like the message block).

The idea is to negate the influence of adding the PRNGs output by choosing a correct S-box output. We can easily find a message block that will make the addition of the two to be equal to 0 modulo 2^8 (we know the previous state of PRNG so we can easily calculate the output of current iteration and choose correct S-box output). So every compression function call will just change 24 most significant bits of every state word.

But not only will it leave the least significant bits unchanged, it will also change all the most significant bit in every word in the same way. The change will simply be the addition of a random 24 bit number.

So when we look at the full state of the function, every iteration is just the addition of a random number to 24 most significant bits of every word.

When we add random numbers modulo 2^{24} we should get the same result once per every $2^{24}+1$ iterations. This results in a collision of the internal state after about 2^{24} iterations.

5.5 Attack on full function

Due to the fact that PRNGs state inflicts the finalization round of the algorithm, we require that this state is the same before we start this round. The same state appears once every $2^{64}-1$ iterations. So we can estimate that we get a collision of both internal states (of the compression function and the PRNGs) in about $2^{24}2^{64} = 2^{88}$ iterations.

5.6 Attack complexity and trade-offs

The attack complexity need to be divided into two parts:

- finding a correct message prefix,
- iterating a function to find a collision.

Complexity of finding a prefix depends only on the desired hash length. We need to have an internal state fulfilling the attack criteria. This means that n internal state words need to have the least significant bytes set to correct values. The probability of setting a single byte is 2^{-8} and as the first byte can be set to any value (only the other n-1 bytes need to be set in accordance) . So the probability of getting a correct internal state by injecting random messages is 2^{-8n}. So for 1024-bit hash size we need about 2^{256} hash function calls to get the correct internal state.

Another way of finding the correct prefix is to use the same idea as in the second part of the attack. First we choose the first 8 bits of the message to set 8 least significant bits of first lane to desired state (any state will do, but as we have full control over it, we can choose the one we like). Then we iterate the function with message blocks chosen in the same way as in the second part of the attack, so that those 8 bits don't change. We do that until the least significant bits of all lanes are set to correct values (presented in prefix section).

Each least significant byte is set with probability 2^{-8} but as soon as it hits the correct value it won't change in next iterations. This means that the complexity of finding the correct prefix can be lowered to 2^8 compression function calls (as every byte is independent), but at the cost of making the message longer. Still the prefix length is nothing compared to postfix length, so this attack is preferred.

This means that the complexity of finding the prefix is either 2^{8n} if we want it to be as short as possible or 2^8 if we can use a longer one. For postfix the complexity is constant and about 2^{88}. So the full attack should need slightly more then 2^{88} compression function calls

This means that this attack can be applied to any hash size longer or equal to 192 bits and easily breaks both the required SHA-3 lengths.

Attacks memory complexity is almost non existent as we need to keep only one state size, and for longer messages we only need to keep their length as we can recreate them with ease.

6 Supposed attack application to StreamHash 3

In the same paper as StreamHash 2, one can find a proposal of even a better version of the algorithm. StreamHash 3 is supposed to be more secure against side channel attacks and faster in parallel implementation than its predecessor.

> The solution for the planned StreamHash3 is to replace S-BOXes with the constructions based on shifts ($\ll$and$\gg$) and modular addition should allow to process input stream word-by-word instead of octet-by-octet, and to implement non-linearity with the SIMD instructions.

Using different octet-by-octet transformation with the rest of the algorithm unchanged leads to the same problem. The flaw that was shown in this paper is based on the untypical parallel construction of the algorithm, not on the underlying S-Box. So unless there are more severe changes to the algorithm, then those involved in this attack should still hold.

Using the bijective word-by-word transformation, it may be trivially easy to keep the selected lane of transformation unchanged. We just need to select such an input to the transformation (which will replace the S-box) of one of the lanes that produces the output which added to the PRNGs output in current round gives us zero. This will lead to the collision-attacks on the internal state with the complexity of 2^{32}.

This attack can also be applied to the full function using the same framework as in the attacks on StreamHash 2, but with much lower complexity as we get a full internal state collision in every iteration. This means that we will get a full function collision in about 2^{64} (PRNGs period).

And this is just a generic attack that can be derived from the hash function design without any knowledge of the transformation itself (we only assume that the transformation is bijective, if not it will just appear after a few rounds, but non bijective

transformation will lead to other problems). If the transformation has some additional features, it may be even easier to mount the attack.

7 Conclusions

We have shown that the StremHash family is still insecure. The changes made between StreamHash 1 and StreamHash 2 do not improve the security of the design to the desired level. Finding a collision is still possible below the birthday bound with the complexity 2^{88} for the hash sizes of m of at least 192 bits.

The round function of the algorithm is totally insecure and it is quite easy to find the internal state fixed points. The method of creating them can lead to new attacks on the algorithm.

The whole security of the algorithm depends on the impact that newly included PRNG has on a finalization function but after some additional research we believe the effect and the attack complexity could be lowered.

But even without any improvements, our attack shows that the StreamHash compressions function construction is not a good starting point to create a good hash function. It sacrifices too much of the security for the ease of algorithm parallelisation.

References

[1] Trojnara M., StreamHash Algorithm Specifications and Supporting Documentation, SHA-3 Submission package (2008).

[2] Trojnara M., Evolution of the StreamHash Hash Function Family, Annales UMCS Informatica 11(2) (2011): 25; DOI: 10.2478/v10065-011-0013-8.