

 Open access • Book Chapter • DOI:10.1007/978-3-642-04474-8_6

Structural Attacks on Two SHA-3 Candidates: Blender-n and DCH-n

— [Source link](#) 

Mario Lamberger, Florian Mendel





Institutions: Graz University of Technology

Published on: 04 Sep 2009 - International Conference on Information Security

Topics: Preimage attack, Collision attack, SHA-2, Hash function and Collision resistance

Related papers:

- [Structural Attacks on Two SHA-3 Candidates: Blender-n and DCH-n](#)
- [Practical \(Second\) Preimage Attacks on the TCS_SHA-3 Family of Cryptographic Hash Functions.](#)
- [Preimages for Reduced SHA-0 and SHA-1](#)
- [Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1](#)
- [Converting meet-in-the-middle preimage attack into pseudo collision attack: application to SHA-2](#)

Share this paper:    

View more about this paper here: <https://typeset.io/papers/structural-attacks-on-two-sha-3-candidates-blender-n-and-dch-59otw44f0v>

Structural Attacks on Two SHA-3 Candidates: Blender- n and DCH- n

Mario Lamberger and Florian Mendel

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria
`mario.lamberger@iaik.tugraz.at`

Abstract. The recently started SHA-3 competition in order to find a new secure hash standard and thus a replacement for SHA-1/SHA-2 has attracted a lot of interest in the academic world as well as in industry. There are 51 round one candidates building on sometimes very different principles.

In this paper, we show how to attack two of the 51 round one hash functions. The attacks have in common that they exploit structural weaknesses in the design of the hash function and are independent of the underlying compression function. First, we present a preimage attack on the hash function Blender- n . It has a complexity of about $n \cdot 2^{n/2}$ and negligible memory requirements. Secondly, we show practical collision and preimage attacks on DCH- n . To be more precise, we can trivially construct a $(2^8 + 2)$ -block collision for DCH- n and a 1297-block preimage with only 521 compression function evaluations. The attacks on both hash functions work for all output sizes and render the hash functions broken.

Key words: Hash functions, collision attacks, preimage attacks, SHA-3, Blender, DCH

1 Introduction

Until 2005, the number of papers on the cryptanalysis of hash functions was quite easy to overlook. This changed significantly with the dawn of the work of Wang *et al.* [25,26]. The weaknesses discovered in MD5 and SHA-1 had wide reaching consequences and were a wake-up call for both academia and industry. The SHA-2 family [18] was only considered to be a temporary solution. Although no full attacks on a member of SHA-2 have been found to date, the fact that the design and security principles are very close to those of SHA-1 raised doubts about the long term security of the SHA-2 family.

As a consequence, the National Institute of Standards and Technology (NIST) has launched a similar competition [19] as it has done for the Advanced Encryption Standard (AES) to replace DES. This time, the goal is to find a new secure hash standard SHA-3. As of now, 51 submissions have advanced to the first round of the SHA-3 competition. Supported by the cryptographic community,

the task of NIST is now to find the best hash function in terms of a wide spectrum of requirements, such as speed and security.

The proposals of round one are based on a great variety of security considerations and design principles. Many of them are block cipher based, using especially AES or parts of AES as building blocks. Some hash functions are based on asymmetric primitives and again others are mere curiosities. From the design perspective, there are Merkle-Damgård [5,17] constructions and variations thereof, sponge constructions [2], HAIFA constructions [3], wide pipe constructions [15], etc.

In this paper, we want to demonstrate vulnerabilities of two designs, namely the hash functions Blender- n [4] and DCH- n [27]. Although they are both based on quite different design principles, they have in common that an attacker can omit the tedious task of going into the details of the respective round transformations. We have identified weaknesses in both design principles.

We present a structural preimage attack on the hash function Blender- n that has a complexity of about $n \cdot 2^{n/2}$ and negligible memory requirements. Furthermore, we show practical collision and preimage attacks on DCH- n and show that we can trivially construct a $2^8 + 2$ -block (multi)-collision for DCH- n and 1297-block preimages with only 521 compression function evaluations.

Our paper underlines the importance of a well founded design principle for a hash function since a bad choice of the iteration mode renders the efforts put in the compression function design ineffectual.

The paper is organized as follows. Section 2 describes our preimage attack on Blender- n . Then, Section 3 demonstrates a practical collision and a practical (second) preimage attack on DCH- n . In Section 4 we conclude.

2 A Preimage Attack on Blender- n

In this section, we present a preimage attack on Blender- n with a complexity of about $n \cdot 2^{n/2}$ and negligible memory requirements. The attack is independent of the compression function of Blender- n and works for all output sizes. A very similar preimage attack for Blender- n was independently proposed in [20]. It has a slightly higher attack complexity of about $n \cdot 2^{(n+w)/2}$, where w is 32 or 64 bits depending on the word size of the hash function. We are well aware of the attacks on Blender- n which concentrate on the internal structure of the compression function presented in [10,11,13]. Even though our attack is less efficient compared to the attack [11], it is superior in the sense that it isn't affected by any tweak to the compression function. Furthermore, due to the generic nature of our attack, it may be applicable to a wider range of hash function designs. For example, the SHA-3 candidate AURORA has been recently broken by similar principles [6,23,24].

2.1 Description of Blender- n

The hash function Blender- n is an iterated hash function. It processes message blocks of 32 (or 64) bits and produces a hash value of 224, 256 (or 384, 512)

bits. If the message length is not a multiple of 32 (or 64) bits, an unambiguous padding method is applied. For the description of the padding method we refer to [4]. Let $W = W_1 \| W_2 \| \dots \| W_t$ be a t -block message (after padding).

In the following \neg denotes the bitwise complement and Σ denotes summation modulo 2^w where w is the wordsize (32 or 64-bit). The hash value h is computed from the chaining values A_i as follows (see Figure 1):

$$h = \Sigma_{i=1}^{t+2} A_i .$$

The chaining values A_i are computed as follows:

$$A_0 = IV \tag{1}$$

$$A_i = f(A_{i-1}, W_i) \quad \text{for } 0 < i \leq t \tag{2}$$

$$A_{t+1} = f(A_t, \Sigma_1) \tag{3}$$

$$A_{t+2} = f(A_{t+1}, \Sigma_2) , \tag{4}$$

where $\Sigma_1 = \neg \Sigma_{i=1}^t W_i$, $\Sigma_2 = \Sigma_{i=1}^t \neg W_i$ and IV is a predefined initial value.

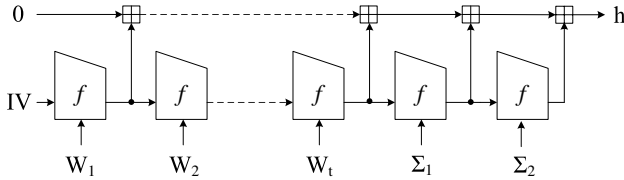


Fig. 1. Structure of the hash function Blender- n .

As can be seen in (3) and (4), Blender- n specifies two checksums (Σ_1 and Σ_2) consisting of the modular addition of all message blocks, which are then input to the two final application of the compression function f . Computing this checksum is not part of most commonly used hash functions such as MD5 and SHA-1.

The compression function f basically consist of 4 steps:

1. Compute the preliminary intermediate values using add-with-carry.
2. Compute the rotation factor r .
3. Rotate the intermediate values.
4. Compute the next state A_i .

For a detailed description of the Blender- n compression function we refer to [4], since we do not need it for our analysis.

2.2 A Preimage Attack on Blender- n

In this section, we present a preimage attack on the hash function Blender- n . It has a complexity of about $n \cdot 2^{n/2}$ and negligible memory requirements. It is based on the following two observations.

Observation 1 *The checksums Σ_1 and Σ_2 are strongly related.*

In other words, the second checksum does not increase the security of Blender- n . This will be very useful for our attack. Let $X = \Sigma_{i=1}^t W_i$ then:

$$\begin{aligned}\Sigma_1 &= \neg \Sigma_{i=1}^t W_i = \neg X \\ \Sigma_2 &= \Sigma_{i=1}^t \neg W_i = \Sigma_{i=1}^t (-W_i - 1) = -t - \Sigma_{i=1}^t W_i = -t - X\end{aligned}$$

Note that $\neg W_i = -W_i + 1$ and hence $\neg W_i = -W_i - 1$.

Observation 2 *The final hash value h of Blender- n is computed from the chaining values A_i by modular additions.*

In other words, the computation of h is invertible. This will be very useful for our attack. Assume, that we can find 2^n messages w^* (and hence chaining values A_i^* for $0 < i \leq t$), such that all produce the same value A_t and X , then we have constructed a preimage for h . This is similar to recent attacks on GOST [16] and Damgård-Merkle hash functions with linear or additive checksums [7].

Based on this short description, we will now show how to find messages w^* which all produce the same value A_t and lead to the same checksum value with a complexity of about $n \cdot 2^{n/2}$ and negligible memory requirements. For the sake of simplicity let $n = 512$ for the remainder of this section. Note that the attack works similar for the other output sizes of Blender- n .

Assume we want to construct a preimage for Blender-512 consisting of 2561 message blocks, *i.e.* $m = W_1 \| W_2 \| \dots \| W_{2561}$. The attack basically consists of two steps and uses multicollisions. It can be summarized as follows.

STEP 1: Constructing the multicollision. A multicollision is a set of messages of equal length that all lead to the same hash value. As shown in [8], constructing a 2^t collision, *i.e.* 2^t messages consisting of t message blocks which all lead to the same chaining value, can be done with a complexity of about $t \cdot 2^{n/2}$ for any iterated hash function.

In the attack we want to construct a 2^{512} collision for the iterative part (chaining values), to get 2^{512} messages w^* (and hence chaining values A_i^*) leading to the same value A_t and X . This has a complexity of about $512 \cdot 2^{288} = 2^{297}$.

However, in the case of Blender- n constructing a multicollision is slightly more complicated. First, due to the small size of the message blocks (64 bits) we need several blocks to construct a collision in the chaining values. Second, to ensure that Σ_1 and Σ_2 (respectively, $X = \Sigma_{i=1}^k W_i$) are equal we need one additional block. In detail, by using 5 message blocks we can construct a collision in the iterative part (chaining values) and the checksums. Since for Blender-512 the chaining value has 512 bits and X has 64 bits, this has a complexity of about 2^{288} using a generic birthday attack.

However, due to the simple structure of the checksum value X , we can easily guarantee that X collides by choosing the message blocks carefully in the attack. It can be summarized as follows:

1. Choose an arbitrary value for d .
2. For all $2^{4 \cdot 64} = 2^{256}$ choices of W_i, \dots, W_{i+3} adjust W_{i+4} accordingly such that $\sum_{j=i}^{i+4} W_j = d$ is fulfilled and compute A_{i+4} with $i > 0$.
3. After computing all 2^{256} candidates for A_{i+4} we expect to find a collision due to the birthday paradox.

In other words, we can find a collision for the iterative part (chaining values) and X with a complexity of about 2^{256} instead of 2^{288} . Furthermore, the memory requirements can be significantly reduced by applying a memory-less variant of the birthday attack [22].

Hence, we can construct a 2^{512} collision with a complexity of about $512 \cdot 2^{256} = 2^{265}$ and negligible memory requirements.

STEP 2: Constructing the preimage for h . In the previous step we constructed a 2^{512} collision in the first $5 \cdot 512 = 2560$ iterations of the hash function. Hence, we have 2^{512} messages w^* leading to the same chaining value A_{2560} and to a collision in X (and hence in the two checksums Σ_1 and Σ_2).

Next we append an additional message block W_{2561} to w^* such that the padding of each of the messages $m^* = w^* \| W_{2561}$ is correct. It is easy to see that appending one message block has no effect on the multicollision in the iterative part and the checksums.

From this set of 2^{512} messages m^* that all lead to the same chaining value A_{2560} and X , we now have to find a message m^* having h as hash value. We write $h = h^* + A_{2561} + A_{2562} + A_{2563}$ where h^* is one of the 2^{512} values:

$$h^* = \sum_{i=1}^{512} (A_{5i-4}^{r_i} + A_{5i-3}^{r_i} + \dots + A_{5i}^{r_i}),$$

with $r_i \in \{0, 1\}$. Here, $(A_{5i-4}^0, A_{5i-3}^0, \dots, A_{5i}^0)$ and $(A_{5i-4}^1, A_{5i-3}^1, \dots, A_{5i}^1)$ are the corresponding 5-block chaining values constituting the multicollision. To find the correct h^* and hence the message leading to the preimage of h we make use of a meet-in-the-middle attack.

First, we save all values for

$$S_1 = \sum_{i=1}^{256} (A_{5i-4}^{r_i} + A_{5i-3}^{r_i} + \dots + A_{5i}^{r_i})$$

in the list L . Note that we have in total 2^{256} values for S_1 in L . Second, we compute

$$S_2 = \sum_{i=257}^{512} (A_{5i-4}^{r_i} + A_{5i-3}^{r_i} + \dots + A_{5i}^{r_i})$$

and check if $h^* - S_2$ is in the list L . After testing all 2^{256} values for S_2 , we expect to find a matching entry in the list L and hence a message w^* that leads to $h^* = S_1 + S_2$. This step of the attack has a complexity of 2^{256} and memory requirements of 2^{256} . Once we have found w^* , we have found a preimage for Blender-512 consisting of $2560+1$ message blocks, namely $m^* = w^* \| W_{2561}$. Note that the memory requirements of the attack can significantly be reduced by applying a memory-less variant of the meet-in-the-middle attack introduced by Quisquater and Delescaille in [22].

Hence, a preimage can be constructed for Blender-512 with a complexity of 2^{265} and negligible memory requirements. Note that in a similar way one can construct preimages for all output sizes of Blender- n with a complexity of about $n \cdot 2^{n/2}$.

3 Practical Collision and Preimage Attacks on DCH- n

3.1 Description of DCH- n

The hash function DCH- n [27], proposed by Wilson, is an iterated hash function based on the Merkle-Damgård design principle and produces a hash value of $n = 224, 256, 384$ or 512 bits. It processes message blocks of 504 bits and preprocesses the input blocks by adding 8 bits of dithering input. At the end, standard MD strengthening is applied.

In each iteration the compression function f is used to update the chaining value of 512 bits as follows:

$$H_{i+1} = f(H_i, M_i) = H_i \oplus M_i \oplus g(M_i), \quad (5)$$

where $g(M)$ is some non-linear transformation. The author of DCH- n claims that the hash function makes use of the Miyaguchi-Preneel mode of operation for block cipher based hash constructions [21]. Nevertheless, a quick look at equation (5) shows that the chaining value H_i is not introduced to the non-linear function g . This fact will be exploited by our attack. For a detailed description of DCH- n we refer to [27].

3.2 Cryptanalysis

In this section, we will present our collision and preimage attack on DCH- n . The attack is an extension of the attack of Khovratovich and Nikolic [9] and is based on similar principles as the attacks on SMASH [12].

A 512-bit block M_i in iteration i consists of $m_i \| M'_i$, where m_i is the 8-bit dithering input and M'_i is the original message block. The 8-bit dithering m_i consists of two parts. The 5 least significant bits are a simple counter, that increments with every iteration. The 3 most significant bits are determined by an encoding of the optimal moves in the “Towers of Hanoi”-sequence, where a new step is generated whenever the 5-bit counter is reset. This sequence is a square-free sequence and therefore assumed to be a good choice for dithering. For closer details, we refer to [27]. At this point, we also want to refer to the work of Andreeva *et al.* [1] that studies the limits on dithering based designs in general.

Let us introduce the function $\gamma_i(M'_i) := g(m_i \| M'_i) \oplus (m_i \| M'_i)$, that is, we combine the XOR from the definition and the dithering corresponding to block i into one function. Then (5) can be rewritten as:

$$H_{i+1} = H_0 \oplus \gamma_0(M'_0) \oplus \gamma_1(M'_1) \oplus \dots \oplus \gamma_i(M'_i) \quad (6)$$

Collision Attack. We now describe the collision attack. We start with a message consisting of $N + 1$ message blocks, $m = M_0 \| M_1 \| \dots \| M_N$. Each $M_i = m_i \| M'_i$, where m_i is computed according to the dithering rule, for $i = 0, \dots, N$. For an 8-bit dithering, there are only 2^8 possible dithering blocks, so if N is large enough, there must be $0 \leq i, j \leq 2^8$ with $i \neq j$ such that $m_i = m_j$ and thus, $\gamma_i = \gamma_j$. Based on (6), we have

$$H_{N+1} = H_0 \oplus \gamma_0(M'_0) \oplus \gamma_1(M'_1) \oplus \dots \oplus \gamma_N(M'_N).$$

So setting $M'_i = M'_j = a \in \{0, 1\}^{504}$ for the above $i \neq j$ implies that these blocks don't contribute to the value H_{N+1} and can thus be freely chosen.

Without looking on the dithering rule for DCH- n , we simply could set $N = 2^8$ to get colliding messages having $2^8 + 2$ blocks (one final block for the padding). Note, that since the ‘‘Towers of Hanoi’’-sequence only has 6 valid states, a smaller choice for N would be $N = 6 \cdot 2^5 = 192$. The bottom line is that we can trivially construct collisions for DCH- n , independently of the concrete dithering method. The messages in the colliding message pair consist of $2^8 + 2$ message blocks.

Every choice of $a \in \{0, 1\}^{504}$ leads to a collision. Hence, we can trivially construct t -collisions (for $0 < t < 2^{504}$) for DCH- n . Note that these attacks apply to DCH- n for all output sizes. Due to size considerations, we don't include an actual colliding message pair.

Preimage Attack. The core observation for the preimage attack is that the outputs of DCH- n form a vector space of dimension n over $GF(2)$. This can be easily seen when looking at the alternative description of DCH- n in (6). A similar approach was used in the attack on the hash function family SMASH- n in [12]. Therefore, the task is to compute a basis of the vector space generated by the DCH- n outputs in order to construct preimages for DCH- n . Again, the only technicality we have to take care of is the dithering of the message blocks.

In the following we assume $n = 512$ since the other output lengths of DCH- n result from truncations of DCH-512. To describe our preimage attack, we will use the following two technical lemmas. As in the collision case we will need to find different indices (i, j) for which the dithering blocks m_i and m_j , and thus, γ_i and γ_j , are the same. For the collision attack we needed only one such index pair whereas for the preimage case this won't suffice. The first lemma will tell us how many message blocks our preimage needs to have to guarantee a certain number of such index pairs.

Lemma 1 *For a message having $N = 2 \cdot \ell + 2^8$ or more message blocks, we can be certain to have at least ℓ index pairs $(i_0, j_0), \dots, (i_{\ell-1}, j_{\ell-1})$ that satisfy $\gamma_{i_k} = \gamma_{j_k}$ for all k and where all occurring indices are unique.*

Proof. We need to guarantee that among all indices from $0, \dots, N - 1$ we can find ℓ pairs as described above. If we take a look at the 8-bit dithering strings m_i for $i = 0, \dots, N - 1$ we know, that the 3 non-counter bits can only have 8 different values $0, 1, \dots, 7$ (actually 6 for the ‘‘Towers of Hanoi’’-sequence). Let

n_0, \dots, n_7 denote the frequencies with which the values $0, \dots, 7$ occur in the non-counter part of the first N dithering messages. To every non-counter block, there correspond 2^5 counter blocks. Thus, $N = 32 \cdot \sum_{i=0}^7 n_i$. From this, the number of sought pairs (i_k, j_k) is

$$32 \cdot \sum_{i=0}^7 \left\lfloor \frac{n_i}{2} \right\rfloor = 32 \cdot \sum_{i=0}^7 \left(\frac{n_i}{2} - \left\{ \frac{n_i}{2} \right\} \right) \geq \frac{N}{2} - 2^7.$$

Therefore, $N = 2 \cdot \ell + 2^8$ is a valid choice of N . \square

The second lemma is concerned with the probability that random vectors from $GF(2)^n$ contain a basis and is a well known result (cf. [14]).

Lemma 2 *The probability for $\ell \geq n$ vectors drawn uniformly at random from $GF(2)^n$, to span a space of dimension n is*

$$\prod_{i=0}^{n-1} \frac{2^\ell - 2^i}{2^\ell} = \prod_{i=0}^{n-1} (1 - 2^{i-\ell}).$$

Now, the attack can be summarized as follows:

1. Assume we want to construct a preimage for h consisting of $N + 1$ message blocks. Thus, we have to find a message M such that:

$$h = H_0 \oplus \bigoplus_{i=0}^N \gamma_i(M_i).$$

2. We choose the last message block M_N such that the padding is correct.
3. Once we have fixed the last message block, we have to find the remaining message blocks M'_i for $0 \leq i < N$ such that:

$$\bigoplus_{i=0}^{N-1} \gamma_i(M'_i) = h \oplus H_0 \oplus \gamma_N(M'_N) \quad (7)$$

4. According to Lemma 1 we choose $N = 2 \cdot \ell + 2^8$ in order to have $\ell \geq 512$ index pairs $(i_0, j_0), \dots, (i_{\ell-1}, j_{\ell-1})$ satisfying $\gamma_{i_k} = \gamma_{j_k}$ (where every i_k, j_k is unique).
5. Next, we compute ℓ vectors $a^k = \gamma_{i_k}(M_0^{k'}) \oplus \gamma_{j_k}(M_1^{k'})$ for $k = 0, \dots, \ell - 1$ with random $M_0^{k'}$ and $M_1^{k'}$ and save the triples $(a^k, M_0^{k'}, M_1^{k'})$ in a list L .
6. From the set of $\ell \geq 512$ vectors a^k we try to compute a basis of the output vector space of DCH- n . If we succeed, this means that we can basically construct such a basis with a complexity of $2 \cdot \ell$ compression function evaluations. This can be reduced to $\ell + 1$ evaluations of the compression function by fixing the block $M_0^{k'}$ and letting only the block $M_1^{k'}$ vary when generating the vectors a^k in the previous step.

Lemma 2 implies that for a choice of $\ell = 520$ we already have a probability of 0.9961 for finding a basis among the a^k and thus need 521 compression function evaluations. Note, that constructing the basis is a one time effort. Let now $\mathcal{B} = \{a^{k_0}, \dots, a^{k_{511}}\}$ denote the basis for the output vector space and let $\mathcal{I} = \bigcup_{k=0}^{511} i_k \cup j_k$ be the union of all the indices contributing to these basis vectors. (For simplicity we assume that the first n pairs correspond to the basis vectors.)

7. We divide the indices $\mathcal{N} = \{0, \dots, N-1\}$ into \mathcal{I} and $\mathcal{N} \setminus \mathcal{I}$. For every index i in $\mathcal{N} \setminus \mathcal{I}$ we set $M'_i = 0 \dots 0$. These message blocks correspond to the indices not contributing to the basis. From (7) we thus get

$$\bigoplus_{\mathcal{I}} \gamma_i(M'_i) = h \oplus H_0 \oplus \gamma_N(M'_N) \bigoplus_{\mathcal{N} \setminus \mathcal{I}} \gamma_i(0 \dots 0).$$

Once a basis \mathcal{B} and the indices \mathcal{I} are computed, the right side of the equation is completely known and thus we have

$$\bigoplus_{\mathcal{I}} \gamma_i(M'_i) = c$$

8. An arbitrary c can be represented with respect to this basis $c = x_0 a^{k_0} + \dots + x_{511} a^{k_{511}}$ by solving the linear system over $GF(2)$. Now we choose the blocks M'_i for $i \in \mathcal{I}$ as follows:
- If $x_k = 0$ for $0 \leq k < n$, we set $M'_{i_k} = \alpha$ and $M'_{j_k} = \alpha$ for some arbitrary value of $\alpha \in \{0, 1\}^{504}$ (as in the collision attack). In this case, γ_{i_k} and γ_{j_k} are equal, these two values cancel out and don't contribute to the result.
 - If $x_k = 1$ for $0 \leq k < n$, we set $M'_{i_k} = M_0^{k'}$ and $M'_{j_k} = M_1^{k'}$ such that $\gamma_{i_k}(M_0^{k'}) \oplus \gamma_{j_k}(M_1^{k'}) = a^k$ for $0 \leq k < n$.

Hence we can construct a preimage by solving a linear system of equations of dimension 512×512 over $GF(2)$. Constructing the basis has a complexity of $\ell + 1$ compression function evaluations and is a one time effort.

Furthermore, the preimage attack can be used to construct second preimages for DCH- n with the same complexity. Note that by using the above described method, preimages (or second preimages) always consist of $N + 1 = 2\ell + 2^8 + 1$ message blocks.

4 Conclusion

In this paper, we were investigating two round one candidates of the SHA-3 hash function competition of NIST. Namely, we were interested in a cryptanalysis of DCH- n and Blender- n by solely investigating the iteration mode.

We showed a preimage attack on the hash function Blender- n for all output sizes. The attack has a complexity of about $n \cdot 2^{n/2}$ compression function evaluations and negligible memory requirements. It is based on structural weaknesses in the design of the hash function and is independent of the compression

function f . Furthermore, we also presented that it is trivial to construct collisions and (second) preimages for DCH- n . The presented attack applies to all similar constructions not introducing the chaining variable into the compression function.

We want to emphasize once more that the main target of the underlying paper was to identify weak design philosophies of hash functions and to learn our lessons from the attacks. It has to be noted that the vulnerabilities pinpointed in this paper are not isolated cases. Our attack on Blender- n has quite some resemblance to the attack on the Russian hash function standard GOST [16] and the recent attack on the SHA-3 candidate AURORA [6,23,24]. The attacks on DCH- n are relying on similar principles as the attacks on the hash function SMASH [12].

Acknowledgements

The authors wish to thank David Wilson and the anonymous referees for useful comments and discussions. The work in this paper has been supported in part by the European Commission under contract ICT-2007-216646 (ECRYPT II) and in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).

References

1. Andreeva, E., Bouillaguet, C., Fouque, P.A., Hoch, J.J., Kelsey, J., Shamir, A., Zimmer, S.: Second preimage attacks on dithered hash functions. In: Smart, N.P. (ed.) EUROCRYPT. LNCS, vol. 4965, pp. 270–288. Springer (2008)
2. Bertoni, G., Daemen, J., Assche, G.V., Peeters, M.: Sponge Functions. ECRYPT Hash Workshop 2007, Barcelona, May 24–25 (2007), <http://events.iaik.tugraz.at/HashWorkshop07>
3. Biham, E., Dunkelman, O.: A Framework for Iterative Hash Functions - HAIFA. Cryptology ePrint Archive, Report 2007/278 (2007), <http://eprint.iacr.org>
4. Bradbury, C.: BLENDER: A Proposed New Family of Cryptographic Hash Algorithms. Submission to NIST (2008), <http://ehash.iaik.tugraz.at/uploads/5/5e/Blender.pdf>
5. Damgård, I.: A design principle for hash functions. In: Brassard, G. (ed.) CRYPTO. LNCS, vol. 435, pp. 416–427. Springer (1989)
6. Ferguson, N., Lucks, S.: Attacks on AURORA-512 and the Double-Mix Merkle-Damgaard Transform. Cryptology ePrint Archive, Report 2009/113 (2009), <http://eprint.iacr.org/>
7. Gauravaram, P., Kelsey, J.: Linear-XOR and Additive Checksums Don't Protect Damgård-Merkle Hashes from Generic Attacks. In: Malkin, T. (ed.) CT-RSA. LNCS, vol. 4964, pp. 36–51. Springer (2008)
8. Joux, A.: Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In: Franklin, M.K. (ed.) CRYPTO. LNCS, vol. 3152, pp. 306–316. Springer (2004)
9. Khovratovich, D., Nikolic, I.: Cryptanalysis of DCH-n (2008), available online: <http://l3j.streamclub.ru/papers/hash/dch.pdf>

10. Klima, V.: A near-collision attack on Blender-256. Available online (2008), http://cryptography.hyperlink.cz/BMW/near_collision_blender.pdf
11. Klima, V.: Huge Multicollisions and Multipreimages of Hash Functions BLENDER-n. Cryptology ePrint Archive, Report 2009/006 (2009), <http://eprint.iacr.org/>
12. Lamberger, M., Pramstaller, N., Rechberger, C., Rijmen, V.: Analysis of the Hash Function Design Strategy Called SMASH. *IEEE Transactions on Information Theory* 54(8), 3647–3655 (2008)
13. Liangyu, X., Ji, L.: Semi-free start collision attack on Blender. *Cryptology ePrint Archive*, Report 2008/532 (2008), <http://eprint.iacr.org/>
14. Lidl, R., Niederreiter, H.: *Finite fields, Encyclopedia of Mathematics and its Applications*, vol. 20. Cambridge University Press, Cambridge, second edn. (1997), with a foreword by P. M. Cohn
15. Lucks, S.: A Failure-Friendly Design Principle for Hash Functions. In: Roy, B.K. (ed.) *ASIACRYPT*. LNCS, vol. 3788, pp. 474–494. Springer (2005)
16. Mendel, F., Pramstaller, N., Rechberger, C., Kontak, M., Szmidt, J.: Cryptanalysis of the GOST Hash Function. In: Wagner, D. (ed.) *CRYPTO*. LNCS, vol. 5157, pp. 162–178. Springer (2008)
17. Merkle, R.C.: One Way Hash Functions and DES. In: Brassard, G. (ed.) *CRYPTO*. LNCS, vol. 435, pp. 428–446. Springer (1989)
18. National Institute of Standards and Technology: FIPS 180-3, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-3. Federal Information Processing Standard (October 2008), available online at: <http://csrc.nist.gov/publications/PubsFIPS.html>
19. National Institute of Standards and Technology: Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. Federal Register Notice (November 2007), available online at: <http://csrc.nist.gov>
20. Newbold, C.: Observations and Attacks on the SHA-3 Candidate Blender. Available online (2008), http://ehash.iaik.tugraz.at/uploads/2/20/Observations_on_Blender.pdf
21. Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: A synthetic approach. In: Stinson, D.R. (ed.) *CRYPTO*. LNCS, vol. 773, pp. 368–378. Springer (1993)
22. Quisquater, J.J., Delescaille, J.P.: How Easy is Collision Search. New Results and Applications to DES. In: Brassard, G. (ed.) *CRYPTO*. LNCS, vol. 435, pp. 408–413. Springer (1989)
23. Sasaki, Y.: A 2nd-Preimage Attack on AURORA-512. *Cryptology ePrint Archive*, Report 2009/112 (2009), <http://eprint.iacr.org/>
24. Sasaki, Y.: A Collision Attack on AURORA-512. *Cryptology ePrint Archive*, Report 2009/106 (2009), <http://eprint.iacr.org/>
25. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) *CRYPTO*. LNCS, vol. 3621, pp. 17–36. Springer (2005)
26. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R. (ed.) *EUROCRYPT*. LNCS, vol. 3494, pp. 19–35. Springer (2005)
27. Wilson, D.A.: The DCH Hash Function. Submission to NIST (2008), available online: http://web.mit.edu/dwilson/www/hash/dch/Supporting_Documentation/dch.pdf