Preimage Attack on Blender

Florian Mendel

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria

Florian.Mendel@iaik.tugraz.at

Abstract. In this paper, we present a preimage attack on the hash function Blender for all output sizes. It has a complexity of about $n \cdot 2^{n/2}$ and negligible memory requirements. The attack is based on structural weaknesses in the design of the hash function and is independent of the underlying compression function.

1 Description of Blender

The hash function Blender 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 [1]. Let $W = W_1 \| W_2 \| \cdots \| W_t$ be a t-block message (after padding). The hash value h is computed form 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.

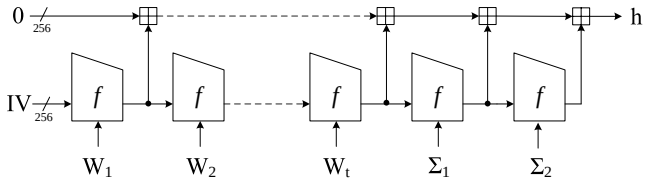

Fig. 1. Structure of the hash function Blender-256.

As can be seen in (3) and (4), Blender 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. 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 compression function we refer to [1], since we do not need it for our analysis.

2 Preimage Attack

In this section, we present a preimage attack on the hash function Blender. 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. This will be very useful for our attack. Let $X = \Sigma_{i=1}^{t} W_i$ then:

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

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

Observation 2 *The final hash value h of Blender 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.

Based on this short description, we will show now 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 $n = 256$ for the remainder of this section. Note the the attack works similar for the other output sizes of Blender.

Assume we want to construct a preimage for Blender-256 consisting of 1281 message blocks, *i.e.* $m = W_1 \| W_2 \| \cdots \| W_{1281}$. The attack basically consists of two steps and uses multi-collisions. It can be summarized as follows.

2.1 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 [2], 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^{256} collision for the iterative part (chaining values), to get 2^{256} messages w^* (and hence chaining values A_i^*) leading to the same value A_t and X. This has a complexity of about $256 \cdot 2^{144} = 2^{152}$.

However, in the case of Blender constructing a multicollision is slightly more complicated. First, due to the small size of the message blocks (32 bits) we need at least 4 message 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-256 the chaining value has 256 bits and X has 32 bits, this hash a complexity of about 2^{144} 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 five message blocks carefully in the attack. It can be summarized as follows:

1. Choose an arbitrary value for d.
2. For all 2^{128} choices of $W_i, \ldots, W_{i+3}$ adjust W_{i+4} accordingly such that $\Sigma_{j=i}^{i+4} W_j = d$ is fulfilled and compute A_{i+4} with $i > 0$.
3. After computing all 2^{128} 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^{128} instead of 2^{144}. Furthermore, the memory requirements can significantly be reduced by applying a memory-less variant of the birthday attack [3].

Hence, we can construct a 2^{256} collision with a complexity of about $256 \cdot 2^{128} = 2^{136}$ and negligible memory requirements.

2.2 Constructing the preimage for h

In the previous step we constructed a 2^{256} collision in the first $5 \cdot 256 = 1280$ iterations of the hash function. Hence, we have 2^{256} messages w^* leading to the same chaining value A_{1280} 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_{1281} to w^* such that the padding of the messages $m^* = w^* \| W_{1281}$ are 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^{256} messages m^* that all lead to the same chaining value A_{1280} and X, we now have to find a message m^* that leads to the the preimage

for $h = h^* + A_{1281} + A_{1282} + A_{1283}$. Note that we have 2^{256} candidates for:

$$h^* = \Sigma_{i=1}^{256}(A_{5i-4}^{r_i} + A_{5i-3}^{r_i} + \cdots + A_{5i}^{r_i})$$

with $r_i \in \{0, 1\}$. To find the correct one, and hence the message leading to the preimage of h we use a meet-in-the-middle attack.

First, we save all values for

$$S_1 = \Sigma_{i=1}^{128}(A_{5i-4}^{r_i} + A_{5i-3}^{r_i} + \cdots + A_{5i}^{r_i})$$

in the list L. Note that we have in total 2^{128} values for S_1 in L. Second, we compute

$$S_2 = \Sigma_{i=129}^{256}(A_{5i-4}^{r_i} + A_{5i-3}^{r_i} + \cdots + A_{5i}^{r_i})$$

and check if $h^* - S_2$ is in the list L. After testing all 2^{128} 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^{128} and a memory requirement of 2^{128}. Once we have found w^*, we have found a preimage for Blender-256 consisting of 1280+1 message blocks, namely $m^* = w^* \| W_{1281}$. 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 [3].

Hence, a preimage can be constructed for Blender-256 with a complexity of 2^{136} and negligible memory requirements. Note that similar attacks can be used to find preimages for all output sizes of Blender.

3 Conclusion

In this paper, we presented a preimage attack on the hash function Blender for all output sizes. The attack has a complexity of about $n \cdot 2^{n/2}$ 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.

References

1. Colin Bradbury. BLENDER: A Proposed New Family of Cryptographic Hash Algorithms. Submission to NIST, 2008.
2. Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *LNCS*, pages 306–316. Springer, 2004.
3. Jean-Jacques Quisquater and Jean-Paul Delescaille. How Easy is Collision Search. New Results and Applications to DES. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *LNCS*, pages 408–413. Springer, 1989.