# Huge multicollisions and multipreimages of hash functions BLENDER-n (draft)

Vlastimil Klima[1)]

**Abstract**
In this paper we present a multicollision and multipreimage attack on the hash function Blender-n for all output sizes n = 224, 256, 384 and 512. The complexity and memory requirements for finding $2^{2n}$ multipreimages (multicollisions) of Blender-n [1] is roughly 10 times more than finding a collision for n/2-bit random hash function.

All previous attacks were based on the trick by Joux [2] using many messages. Our attacks are based on one message with several fixpoints. The state register has eight words. By properly choosing message words we force half of the register to go to the original state. Then we will find a collision in the rest with complexity $2^{n/4}$. The collision creates a fix point in the sequence of states of the state register. We use 10 such fix points.

Previously known attacks [4, 5] on Blender-n have the complexity at least $2^{n/2}$. Our $2^{2n}$-multicollision and multipreimage attacks have a complexity $10*2^{n/4}$.

# 1 An overview of Blender-n

For the sake of simplicity we will only consider Blender-256. The attacks on other variants are similar.

The hash function Blender is an iterated hash function. It uses w-bit words (w = 32 for Blender-256, w = 64 for Blender-512), a state register A of eight w-bit words, two carry bits c1, c2 and a hash register H of eight w-bit words. In the beginning the register H and bits c1, c2 are zeroed. The initial value of the state register A is $A^0 = (a0^0, a1^0, a2^0, a3^0, a4^0, a5^0, a6^0, a7^0) = H_{init}$. The register H contains a chaining value, which is the sum (modulo $2^{32}$ by words) of the states of the state register A, $H^t = \sum_{t=1,...,K} A^t$. A new state is the function of the old state and a message word. So we have $(A^{t+1}, c1^{t+1}, c2^{t+1}) = f(A^t, c1^t, c2^t, W^t)$, where f is the compression function and $W^t$ is the current message word.

The sequence of words $W^t$ is prepared by taking the message and the tail, consisting of fill bytes (fill consists of 13 first bytes repeated to the required length), bit-length of the message, length of the bit-length in bytes and finally two w-bit checksums.

The checksums are computed from all message words (before the first checksum):
checksum1 = non( $\sum_{t=1,...,K} W^t$ ),
checksum2 = $\sum_{t=1,...,K}$ (non$W^t$).
The bit-length of the message is not limited.

---

[1)] Independent cryptologist, Prague, Czech Republic, http://cryptography.hyperlink.cz, v.klima@volny.cz

To avoid technical details, we will assume only messages with integer number of words, having the first 13 bytes the same (for filling), the same length and even the same checksums. It is important to note that checksums, updates of the register H and the register A, are computed from w-bit words, especially the addition is made modulo $2^w$.

Here we remind a part of the original description of the hashing, paragraph 2.6.2, [1]:

The 256-bit algorithm uses eight 32-bit working variables, a0 to a7, eight 32-bit result variables, H0 to H7, and two single-bit carry variables, c1 and c2; these constitute the "state" of the algorithm carried from round to round. This algorithm also uses three 32-bit intermediate values, T, T1 and T2, and one intermediate integer value r used to hold a rotation factor.

Before hash computation begins, the working variables, a0 to a7, are initialized to the following eight 32-bit words ($H_{init}$) in hex: a0 = 6a09e667, ...(cut)..., a7 = 5be0cd19.

After the message has been prepared and the variables initialized, perform the following computations for each 32-bit word $W^t$ in the prepared message:

1. Compute the preliminary intermediate values using add-with-carry:
[c1,T1] = ( a5 ⊕ $W^t$ ) + ( a1 ⊕ rotl(a3, 8) ) + c1
[c2,T2] = ( a0 ⊕ rotr($W^t$, 8) ) + ( a4 ⊕ rotr(a2, 8) ) + c2

2. Compute the rotation factor:
r = 8 – (c1 + c2)

3. Rotate the intermediate values:
T1 = rotl(T1, r)
T2 = rotr(T2, r)

4. Compute the next state:
T = rotr(a0, 7)
a0 = a1 ⊕ T2
a1 = a2 ⊕ T1
a2 = a3 ⊕ T2
a3 = a4 ⊕ T1
a4 = a5 ⊕ T2
a5 = a6 ⊕ T1
a6 = a7 ⊕ T2
a7 = T ⊕ T1

5. Update the hash result variables:
H0 = H0 + a0
H1 = H1 + a1
H2 = H2 + a2
H3 = H3 + a3
H4 = H4 + a4
H5 = H5 + a5
H6 = H6 + a6
H7 = H7 + a7

These five steps constitute one round of the algorithm. After repeating these steps for each word in the prepared message, the resulting 256-bit message digest of the message M is H0 || H1 || H2 || H3 || H4 || H5 || H6 || H7.

# 2 The state register

The hashing process has an internal state, defined by values (A, c1, c2, H). We will create collisions in the state register A and then in the hash register H. For the sake of simplicity we will talk about the state register A, but all of the following computations are made for the state A including the carry bits c1, c2.

The state register has eight words. By a careful choice of 256 message words $W^t$ we will force the state register to have only 4 changing words after every 256th round. Let us denote $A^0 = (a0^0, a1^0, a2^0, a3^0, a4^0, a5^0, a6^0, a7^0) = H_{init}$ the initial value of the state register A.

**The basic attack**

Round 0:

From values of the state $A^0$ we compute the first word $W^0$ of the message so that $\mathbf{T2^0 = 0}$. There is exactly one corresponding value $W^0$, as we can see from the equation T2 = (a0 ⊕ rotr($W^t$, 8)) + (a4 ⊕ rotr(a2, 8)) + c2. Recall that the register A rotates its words one position to the left and at the last word it rotates bits by 7 positions to the right:

$A^1 = ($ **$a1^0$** $, a2^0 \oplus T1^0,$ **$a3^0$** $, a4^0 \oplus T1^0,$ **$a5^0$** $, a6^0 \oplus T1^0,$ **$a7^0$** $, rotr(a0^0,7) \oplus T1^0)$.

Round 1:

Similarly in the round 1 we choose the word $W^1$ of the message such that $\mathbf{T1^1 = 0}$. There is exactly one corresponding value computed from the equation T1 = (a5 ⊕ $W^t$) + (a1 ⊕ rotl(a3, 8)) + c1.

We get

$A^2 = (a2^0 \oplus T1^0 \oplus T2^1,$ **$a3^0$** $, a4^0 \oplus T1^0 \oplus T2^1,$ **$a5^0$** $, a6^0 \oplus T1^0 \oplus T2^1,$ **$a7^0$** $, rotr(a0^0,7) \oplus T1^0 \oplus T2^1,$ **$rotr(a1^0,7)$** $)$.

Round 256:

In the previous rounds the words T2 and T1 are chosen so that they have no influence on values a1, a3, a5, a7. Therefore the state register returns to its original value on odd positions after 256 rounds:

$A^{256} = (a0^{256},$ **$a1^0$** $, a2^{256},$ **$a3^0$** $, a4^{256},$ **$a5^0$** $, a6^{256},$ **$a7^0$** $)$.

# 3 Collisions in the state register

For the sake of simplicity of the filling process (as a part of completing any message before hashing), we assume the first 13 words to be constant. They can be followed by an arbitrarily chosen sequence of bytes, so that we get an integer number of w-bit words at the beginning, for instance 256 words. Let us call this part of the message as the first stationary part (S1).

We start from the last state and then we use the method described above to create states $A^{256*1}$, $A^{256*2}$, $A^{256*3}$, ..., until we find a collision in this sequence. This collisions creates a fix point (or a cycle), because we can return back (and make several cycles) or go on. After the first cycle we make 256 steps with randomly chosen message words $W^t$ (to break the

deterministically defined cycle). Then we continue again with the method from previous section and find the second collision in a new part of the A-states.

Complexity of finding the first and the second collision is $2^{n/4}$ from birthday paradox ($2^{n/4+1}$ including carry bits).

Let us denote S1 the first stationary part of the sequence, C1 the part between the first colliding points, S2 the second stationary part (256 random steps) and C2 the part between the second colliding points.

Note that we can go through the part C1 (C2) as many times as we want.

# 4 Collisions using two fixpoints

Here we will describe one possibility how to use only two fix points. In the next section we will use 10 fixpoints. Now we define two colliding messages M1 and M2.

The first message M1 goes through the part S1 once, then N1 times through the cycle C1, once through the part S2 and once through the cycle C2.

The second message M2 goes once through the part S1, once through the cycle C1, once through the part S2 and N2 times through the cycle C2.

Let us denote L(S1), L(C1), L(S2), L(C2) the number of rounds in appropriate parts S1, C1, S2, C2 of the state sequence. Let L(M1), L(M2) denote the number of rounds of messages M1 and M2.

The hash value is defined as a sum (separately in 8 words, modulo $2^{32}$) of corresponding states A obtained from processing all words of the message.

So, let us denote S(S1), S(C1), S(S2) and S(C2) the contribution of A-states in parts S1, C1, S2, C2, to the hash value (sum). Let S(M1), S(M2) be the whole sum of the states, when processing the entire messages M1 and M2.
Also, let us denote s(S1), s(C1), s(S2) and s(C2) the sums of message words in corresponding parts of the state sequence.

Let us define $N1 = 2^w * L(C2) + 1$, $N2 = 2^w * L(C1) + 1$. Then two messages will have the same lengths, checksums and chaining values (sums of A-states).

**The lengths**
The lengths of the messages M1 and M2 are
$L(M1) = L(S1) + Nl*L(C1) + L(S2) + 1*L(C2) = L(S1) + (2^w * L(C2) + 1)*L(C1) + L(S2) + L(C2) = L(S1) + L(C1) + L(S2) + L(C2) + 2^w * L(C2)*L(C1)$,

$L(M2) = L(S1) + 1*L(C1) + L(S2) + N2*L(C2) = L(S1) + L(C1) + L(S2) + (2^w * L(C1) + 1)* L(C2) = L(S1) + L(C1) + L(S2) + L(C2) + 2^w * L(C2)*L(C1)$,

so the lengths of the messages (in words) are the same $L = L(M1) = L(M2)$.

**The checksums**
Let us denote K the length of a message in words and X the sum of words of a message, $X = \sum_{t=1,...,K} W^t$. Then we have (modulo $2^w$)
$checksum1 = non( \sum_{t=1,...,K} W^t ) = non\ X = 0xFF...FF - X = 1 - X$,
$checksum2 = \sum_{t=1,...,K} (nonW^t) = \sum_{t=1,...,K} (1 - W^t) = K - \sum_{t=1,...,K} W^t = K - X$.

When the messages M1 and M2 have the same length L(M1) and L(M2) and the same sum of all words X(M), then they have also the same checksums checksum1 and checksum2.

Because the sum is computed modulo $2^w$, we have
$X(M1) = s(S1) + Nl*s(C1) + s(S2) + 1*s(C2) = s(S1) + (2^w * L(C2) + 1)*s(C1) + s(S2) + s(C2) = s(S1) + s(C1) + s(S2) + s(C2)$,

$X(M2) = s(S1) + 1*s(C1) + s(S2) + N2*s(C2) = s(S1) + s(C1) + s(S2) + (2^w * L(C1) + 1)*s(C2) = s(S1) + s(C1) + s(S2) + s(C2)$,

so the checksums of messages M1 and M2 are the same.

**The chaining values**
Both messages M1 and M2 end in the last state of the cycle C2. Let us compute their chaining values h(M1) and h(M2).

Because the sums are computed from words modulo $2^w$, we have
$h(M1) = S(S1) + Nl*S(C1) + S(S2) + 1*S(C2) = S(S1) + (2^w * L(C2) + 1)*S(C1) + S(S2) + S(C2) = S(S1) + S(C1) + S(S2) + S(C2)$,

$h(M2) = S(S1) + 1*S(C1) + S(S2) + N2*S(C2) = S(S1) + S(C1) + S(S2) + (2^w * L(C1) + 1)*S(C2) = S(S1) + S(C1) + S(S2) + S(C2)$,

so chaining values are the same as well.

**The hash values**
Note that now we can choose an arbitrary suffix and append it to both messages. Then we complete the hashing by processing the common part of the messages: the filling, length, length of the length and the checksums. We will obtain full collision.

**The complexity and memory requirements**
The complexity and memory requirement for finding above collisions for Blender-n is roughly the same as finding the collision for n/2-bit random hash function.

# 5 Multipreimages

Let us choose a hash value H. We will create message M (in fact huge number of messages) such that h(M) = H in the following steps:

1. Set the first stationary part S1 to a random value (greater then 13). Use the procedure from the section 3 and find the first collision cycle C1. Follow the cycle C1 with a random stationary part S2 (having a very small random size) and the second cycle C2, stationary part S3 (having a very small random size), ..., and finish with the cycle S10 and C10. Let us denote $A^{fin}$ the final state of the state register.
2. Let $A^{fin}$ is the final state of all assumed messages. Note that $A^{fin}$ is the final state after processing all message words (before processing the tail).
3. Let us choose a bit-length L of the message (multipreimages, multicollisions), for instance around the value $w*2^{n/2 + 8+w+log(10)}$ (i.e. $2^{n/2 + 8+w+log(10)}$ w-bit words). All the assumed messages will have the prescribed bit-length L.

4. Let us choose a value X, the future sum of message words. All messages will have the same sum X. Also, all messages will have the same tail (filling, bit-length, length of length, checksums).
5. Knowing the final state $A^{fin}$, the filling, the bit-length, the length of length and checksums, we can process all these words (the tail) and obtain the value dH of their contribution to the hashing register.
6. Now we can "cut" the contribution of the tail from the bit-length, the sum of words and the hash. We obtain adjusted values H, L and X. We can assume that L is directly the number of w-bit words instead of bits (in the following).
7. The task is to find a message (messages) with the length L, the sum of message words X, the final state $A^{fin}$ and the chaining value H.

**Construction of preimages**
For every i = 1, ..., 10 let us denote

L(Ci) - the length of the cycle Ci (in w-bit words),
s(Ci) - the sum of words $W^t$, associated with the states of the cycle Ci
S(Ci) - the sum of states $A^t$, associated with the states of the cycle Ci

L(Si) - the length of the cycle Si (in w-bit words),
s(Si) - the sum of words $W^t$, associated with the states of the cycle Si
S(Si) - the sum of states $A^t$, associated with the states of the cycle Si

We can create a huge number of messages M such that we will go once through the stationary parts S1, ..., S10 and differently many times through the cycles C1, ..., C2. We only need to set the numbers so that the sum of words, the sum of states and the sum of partial lengths will be the same. For every i = 1, ..., 10 let us denote $k_i$ the number of passes through the cycle Ci.

We need to solve the system of equations:

(H): $H = \sum_{i=1,...,10} k_i * S(C_i) \bmod 2^w$
(X): $X = \sum_{i=1,...,10} k_i * s(C_i) \bmod 2^w$
(L): $L = \sum_{i=1,...,10} k_i * L(C_i)$

Note that the equation (H) is a system of 8 equations, because X and $S(C_i)$ are 8-word vectors, (H) is one equation and (L) is also one equation. So there are 10 equations with 10 unknowns $k_i$. If the equation (L) was also modular (mod $2^w$) then we could solve the system simply by Gauss elimination method. But the last equation is without modulo reduction, so we will deal with it more.

**Note.** If we choose more cycles (fixpoints), we have more degrees of freedom in the above system H-X-L (more variables then equations) and therefore we can obtain much more solutions. This can be done for instance by increasing the working factor from 10 to 11. Also, if the system of equations is linearly dependent, we can exclude the cycle, which creates the dependency, and add a new one.

**Solving the system H-X-L**
Let us denote the low and high part of a variable V as $V^L = V \bmod 2^w$, $V^H = (V - V^L)/2^w = V >> w$. Note that
$S(C_i) = S(C_i)^L$ and $s(C_i) = s(C_i)^L$, while $L(C_i) = L(C_i)^H * 2^w + L(C_i)^L$ and $k_i = k_i^H * 2^w + k_i^L$.

We can rewrite H-X-L as
(H): $H = \sum_{i=1,...,10} k_i^L * S(C_i)^L \mod 2^w$
(X): $X = \sum_{i=1,...,10} k_i^L * s(C_i)^L \mod 2^w$
(LL): $L^L = \sum_{i=1,...,10} (k_i^H *2^w + k_i^L)* (L(C_i)^H*2^w + L(C_i)^L) \mod 2^w$
(LH): $L^H = ( \sum_{i=1,...,10} (k_i^H *2^w + k_i^L)* (L(C_i)^H*2^w + L(C_i)^L) ) >> w$

The last two equations are
(LL): $L^L = \sum_{i=1,...,10} k_i^L * L(C_i)^L \mod 2^w$
(LH): $L^H = (\sum_{i=1,...,10} ( k_i^H *2^w *L(C_i) + k_i^L* L(C_i)^H*2^w + k_i^L*L(C_i)^L) ) >> w$
$= c^{2)} + \sum_{i=1,...,10} ( k_i^H *L(C_i) + k_i^L* L(C_i)^H )$

Now we can find solution of the system H-X-LL of 10 linear equations with 10 unknown integer variables $k_i^L \pmod{2^w}$
(H): $H = \sum_{i=1,...,10} k_i^L * S(C_i)^L \mod 2^w$
(X): $X = \sum_{i=1,...,10} k_i^L * s(C_i)^L \mod 2^w$
(LL): $L^L = \sum_{i=1,...,10} k_i^L * L(C_i)^L \mod 2^w$

After that we replace $k_i^L$ in the remaining equation (LH), and we have

(LH): $L^H = \sum_{i=1,...,10} ( k_i^H *L(C_i) + k_i^L* L(C_i)^H ) = \sum_{i=1,...,10} ( k_i^H *L(C_i) ) + CC$
where CC is a constant with the value $\sum_{i=1,...,10} k_i^L* L(C_i)^H$.

It remains to solve one equation with 10 unknown integer variables $k_i^H$. Recall that this is a linear diophantine equation with an exception that we are looking for nonnegative solutions. We will solve it by brute force[3)].

The cycles $C_i$ will contain around $2^{n/4}$ points. The points are states after processing 256 words, so the constants $L(C_i)$ will be around $2^{n/4} * 2^8$ and CC will be around $10 * 2^w * 2^{n/4} * 2^8 \leq 2^{\log(10)+w+n/4+8}$. The value $L^H$ is around $2^{n/2+8+w+\log(10)}/2^w = 2^{n/2 + 8+\log(10)}$, so the value $L^{rest} = L^H - CC$ will also be around $2^{n/2 + 8 +\log(10)}$. We have
(LH): $L^{rest} = \sum_{i=1,...,10} k_i^H *L(C_i)$.
Let us say that the cycle $C_1$ is the smallest one. Now we can set 9 variables $k_i^H$ for i = 2, ..., 10 arbitrarily and then compute $k_1^H$ from the equation (LH). If the expression $L^{rest} - \sum_{i=2,...,10} k_i^H *L(C_i)$ is divided by $L(C_1)$, we get one solution. The solution is found with probability $1/L(C_1)$. Since the difference between $L^{rest}$ and $L(C_i)$ is big, we can expect a huge number of solutions, more than around $((2^{n/2 + 8+\log(10)} /10) / 2^{n/4} * 2^8)^9 / L(C_1) \geq 2^{2n}$.

Every solution represents a way from the initial state, different times passing through 10 fixpoints and finishing in the last known unique state. All these messages have the same prescribed hash value.

The complexity and memory requirements for finding $2^{2n}$ multipreimages (multicollisions) of Blender-n is roughly 10 times more than finding a collision for n/2-bit random hash function.

---

[2)] c is a carry from the low part, it can acquire at maximum 10 possible values, so we can omit it from further considerations and assume c = 0.
[3)] certainly there are more effective methods

# 6 Conclusion

We showed a multicollision and multipreimage attack on the hash function Blender-n for all output sizes. Our $2^{2n}$-multicollision and multipreimage attacks have a complexity $10*2^{n/4}$.

# 7 References


[1] Colin Bradbury: BLENDER, A Proposed New Family of Cryptographic Hash Algorithms, http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/documents/Blender.zip

[2] Antoine Joux: Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions, CRYPTO 2004, LNCS, Vol. 3152, pp. 306-316. Springer, 2004

[3] Vlastimil Klima: A near-collision attack on BLENDER, http://cryptography.hyperlink.cz/BMW/near_collision_blender.pdf

[4] Florian Mendel: Preimage Attack on Blender, http://ehash.iaik.tugraz.at/uploads/4/48/Blender-preimage.pdf

[5] Craig Newbold: Observations and Attacks On The SHA-3 Candidate Blender, http://ehash.iaik.tugraz.at/uploads/4/48/Blender-preimage.pdf

[6] Liangyu Xu: Semi-free start collision attack on Blender, http://ehash.iaik.tugraz.at/uploads/4/48/Blender-preimage.pdf