# Collisions for variants of the BLAKE hash function

Janoš Vidali[a], Peter Nose[a], Enes Pašalić[b]

[a] *University of Ljubljana, FRI, Ljubljana, Slovenia,*
*e-mail: {janos.vidali, peter.nose}@fri.uni-lj.si*
[b] *University of Primorska, FAMNIT, Koper, Slovenia, e-mail: enespasalic@yahoo.se*

**Abstract**

In this paper we present an attack to the BLOKE and BRAKE hash functions, which are weakened versions of the SHA-3 candidate BLAKE. In difference to BLAKE, the BLOKE hash function does not permute the message words and constants in the round computation of the compression function, and BRAKE additionally removes feedforward and zeroes the constants used in each round of the compression function. We show that in these cases we can efficiently find, for any intermediate hash value, a fixed-point block giving us an internal collision, thus producing collisions for messages of equal length in case of BLOKE, and internal collisions for BRAKE.

*Keywords:* BLAKE, BLOKE, BRAKE, collision, cryptography, fixed point, hash functions

## 1. Introduction

Cryptographic hash functions are primitives of fundamental importance for a wide range of applications such as: efficient digital signature algorithms, MAC, integrity check, data origin authentication etc. Though during the last three decades there has been a very active research on this topic, most of the hash function proposals have been broken (MD4, MD5, SHA-0, SHA-1, ...) or they are considered insecure for providing a long-term security. An attempt to increase the trust in hash functions was initiated by NIST in 2007 through an open public competition [11] where the candidate hash functions are supposed to go through stringent public scrutiny before recognizing the proposals as a new SHA-3 hash function.

The cryptographic strength of a hash function is commonly evaluated through the resistance to collision, preimage and second preimage attacks [10]. For an $n$-bit hash value any collision attack beyond the computational effort corresponding to $2^{n/2}$ hash computations, and any preimage or second preimage in less than $2^n$ hash computations compromises the particular design. Exhibiting collisions for a given hash function essentially means that the hash value for two different messages is the same, that is $H(IV, m_1) = H(IV, m_2)$. Note that there might be some additional inputs used in the computation, most standard ones being salt and counter.

The BLAKE hash function family [2] by Aumasson, Henzen, Meier and Phan is an SHA-3 proposal to the NIST Hash Competition. It is one of the 14 submissions that has advanced to the second round of the contest, without any attacks to the full version known yet [1]. BLAKE is built upon the LAKE hash function [4], with HAIFA [7] as the iteration mode and a local wide-pipe as its internal structure, and the ChaCha stream cipher [6] as its core function.

In this paper we present attacks on BLOKE and BRAKE, weakened versions of BLAKE that were left by the designers as cryptanalytic challenges. To find collisions, our method (with a negligible amount of computation) identifies a message block such that the round function does not change the internal state when this message block is hashed. Such message blocks are then used to obtain an intermediate hash value that does not depend on the previous hash value in the case of BLOKE, and to generate an internal collision for BRAKE. The attacks given here confirm the necessity of the design criteria of BLAKE that are removed in BLOKE, namely the use of permutations of the message words and the use of constants which are XORed to the message words, as well as the standard requirement to include the message length when hashing the message.

The remainder of the paper is organized as follows. In Section 2, the design of the BLAKE hash function and its weakened versions together with the related cryptanalytic results are given. The collisions for weakened versions of BLAKE based on a fixed point attack are found in Section 3. Finally, Section 4 concludes the article.

### *1.1. Related work*

In [9], Li and Xu present attacks on BLAKE reduced to 1.5, 2 and 2.5 rounds. Their method is to control one or two words of the intermediate hash value by modifying the message and initial hash value. In the case of 1.5 rounds, only the message is modified to control two words of the hash value, thus obtaining a $2^{160}$ (second) preimage attack and a $2^{80}$ collision attack on BLAKE-32. For 2 and 2.5 rounds, the initial hash value also has to be modified, so the complexities of free-start (second) preimage attack and free-start collision are $2^{224}$ and $2^{112}$, respectively. Hence, a (second) preimage attack on 2 or 2.5 round BLAKE-32 has complexity $2^{241}$.

Another result, due to Guo and Matusiewicz [8], gives us near-collisions (216 bits of the hash) for 4 intermediate rounds of BLAKE-32 with a complexity of $2^{56}$. They achieve this by linearizing the $G_i$ function and introducing differentials of the form `0x88888888` by changing the message, the intermediate hash value, the salt and the counter, making sure that no differences are passed through the rotations by 7.

On the rump session of ASIACRYPT 2009 [12], Wang, Ohta and Sakiyama presented free-start preimage attacks on 4 and 4.5 rounds of BLAKE, with complexities $2^{224}$ and $2^{252}$, and $2^{32}$ and $2^{8}$ memory, respectively.

A recent result due to Aumasson et al. [5], presented at FSE 2010, establishes differential properties of the permutation $G_i$ used in the compression function of BLAKE. An efficient algorithm to invert one round and an improved algorithm to find a preimage of 1.5 rounds of BLAKE-32 with complexity $2^{128}$ are given. The former algorithm is what the attack in this paper is based on. Furthermore, Aumasson et al. exploit the differential properties of $G_i$ to find large classes of impossible differentials for one and two rounds, and some specific cases for five and six rounds of BLAKE, and also find near-collisions for the compression function reduced to four rounds.

## 2. The design of BLAKE

The BLAKE family contains four hash functions: BLAKE-28, BLAKE-32, BLAKE-48, BLAKE-64, with the bit lengths of their digests being 224, 256, 384, and 512, respectively. The former two operate on 32-bit words, while the latter two work with 64-bit words.

### *2.1. BLAKE-32 and BLAKE-64*

To hash a message $M$ with the BLAKE-32 hash function, the message is first padded with at least 66 bits, so that the length of the padded message is a multiple of 512. The last 64 bits of the padding are a binary representation of the bit length of the unpadded message. The message is then split into 512-bit blocks and iteratively input to the compression function together with the previous hash value, a 64-bit counter of bits hashed so far, and an optional 128-bit salt. The previous hash value to the first block is the initialization vector (IV) specified in the documentation. The counter is set to zero in the last block in the case it only consists of padding.

The compression function $\mathit{compress}(h, m, s, t)$ consists of three stages: initialization, round iteration and finalization. The initialization establishes the internal state $v$:

$$v = \begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} = \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ s_0 \oplus c_0 & s_1 \oplus c_1 & s_2 \oplus c_2 & s_3 \oplus c_3 \\ t_0 \oplus c_4 & t_0 \oplus c_5 & t_1 \oplus c_6 & t_1 \oplus c_7 \end{pmatrix}$$

After the initialization, the function $\mathit{round}_r(v, m)$ is called ten times, for $r = 0, \ldots, 9$. This function consists of eight calls of the function $G_i$, which operates on the words of $v$:

$$\begin{array}{llll} G_0(v_0, v_4, v_8, v_{12}) & G_1(v_1, v_5, v_9, v_{13}) & G_2(v_2, v_6, v_{10}, v_{14}) & G_3(v_3, v_7, v_{11}, v_{15}) \\ G_4(v_0, v_5, v_{10}, v_{15}) & G_5(v_1, v_6, v_{11}, v_{12}) & G_6(v_2, v_7, v_8, v_{13}) & G_7(v_3, v_4, v_9, v_{14}) \end{array}$$

The function $G_i(a, b, c, d)$ is a modified version of the stream cipher ChaCha [6]:

$$\begin{array}{rcl}
a & \leftarrow & a + b + (m_{\sigma_r(2i)} \oplus c_{\sigma_r(2i+1)}) \\
d & \leftarrow & (d \oplus a) \ggg 16 \\
c & \leftarrow & c + d \\
b & \leftarrow & (b \oplus c) \ggg 12 \\
a & \leftarrow & a + b + (m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)}) \\
d & \leftarrow & (d \oplus a) \ggg 8 \\
c & \leftarrow & c + d \\
b & \leftarrow & (b \oplus c) \ggg 7
\end{array}$$

Here, $c_j$ for $j = 0, \ldots, 15$ are constants, and $\sigma_r$ for $r = 0, \ldots, 9$ are permutations of $\mathbb{Z}_{16}$. Both are defined in the BLAKE documentation. Finally, the output hash value $h'$ is computed:

$$h'_i \leftarrow h_i \oplus s_{i \bmod 4} \oplus v_i \oplus v_{i+8} \text{ for } i = 0, \ldots, 7.$$

The main difference between BLAKE-32 and BLAKE-64 is that the latter uses 64-bit words instead of 32-bit words. As a consequence the length of the message block, internal state, hash values, salt, counter and constants are doubled. Two other important differences are the number of rounds, which is raised to 14, and the rotations in the $G_i$ function, which are by 32, 25, 16 and 11 bits in BLAKE-64. For further details the interested reader is referred to the specification of BLAKE [2]. A description of shortened versions of BLAKE-32 and BLAKE-64, named BLAKE-28 and BLAKE-48 respectively, is also found there.

### *2.2. Weakened versions*

The authors of BLAKE have also published a document describing weakened versions of BLAKE for cryptanalysis [3]. The first one, called BLOKE, substitutes all the permutations $\sigma_r$ with an identity permutation, thus making the function $round_r$ independent of $r$. As a direct consequence, BLOKE can also be seen to have zero constants in the $G_i$ function if we reencode the message by XORing the appropriate constant to each message word. This is also the version our main attack works on.

In FLAKE, the feedforward is removed from the finalization of the compression function, which is then only:

$$h'_i \leftarrow v_i \oplus v_{i+8} \text{ for } i = 0, \ldots, 7.$$

In BLAZE, the constants $c_j$, $j = 0, \ldots, 15$ are all set to zero in the $G_i$ function, and finally, all three changes are made in BRAKE.

## 3. Collisions for weakened versions of BLAKE

Throughout the rest of the paper the following notations are used. When considering a single $G_i$ function, $a, b, c, d$ will denote its input values, $a', b', c', d'$ the values they take when they are first assigned a new value, and $a'', b'', c'', d''$ their final values. In other words, these variables satisfy the following equations in the case of BLAKE-32:

$$\begin{aligned}
a' &= a + b + (m_{\sigma_r(2i)} \oplus c_{\sigma_r(2i+1)}) \\
d' &= (d \oplus a') \ggg 16 \\
c' &= c + d' \\
b' &= (b \oplus c') \ggg 12 \\
a'' &= a' + b' + (m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)}) \\
d'' &= (d' \oplus a'') \ggg 8 \\
c'' &= c' + d'' \\
b'' &= (b' \oplus c'') \ggg 7
\end{aligned}$$

Here, $+$ denotes the 32-bit addition, $\oplus$ the 32-bit bitwise XOR operation, and $\ggg$ the 32-bit cyclical rotation towards the least significant bits: the number 3 is represented as `0x00000003` in the hexadecimal notation, so `0x00000003` $\ggg 1 =$ `0x80000001`. The operators are defined similarly for the 64-bit case.

The / operator is used for integer division, so $x = y/z \Leftrightarrow x = \left\lfloor \frac{y}{z} \right\rfloor$. The mod operator is standard: if $x = y \bmod z$, then $x \in \mathbb{Z}_z$ and $x \equiv y \pmod z$.

If $m$ is a message block, then $m_i$, $0 \le i \le 15$ represents its $i$-th word. Similarly, $v_i$, $0 \le i \le 15$ is the $i$-th word of the state $v$, and $h_i$, $0 \le i \le 7$ is the $i$-th word of the hash value $h$.

Let $round_r$ be the round function for the $r$-th round. Then we can write $v'' = round_r(v, m)$, where $v$ is the state before the current round, $v''$ the state after it and $m$ the message block being hashed. In the BLAKE documentation [2] it is shown that for a fixed message $m$ the $G_i$ functions are bijective. It follows then that also the $round_r(\cdot, m)$ function is bijective and thus a permutation of $\{0,1\}^{512}$. In [5], the same is shown for $round_r(v, \cdot)$ – so there is a unique message input that will transform a fixed initial state into a fixed final state. We give the equations needed to compute the message block used as the input to the compression function, given the initial and final states of the $r$-th round:

$$\begin{aligned}
b &= (c'' - d'') \oplus (c'' \ggg 20) \oplus (b'' \ggg 13) && (1) \\
c &= c'' - d'' - (a'' \oplus (d'' \ggg 24)) && (2) \\
d'' &= c'' - (b \oplus (c'' \ggg 20) \oplus (b'' \ggg 13)) && (3) \\
a'' &= (c'' - d'' - c) \oplus (d'' \ggg 24) && (4) \\
a' &= d \oplus (a'' \ggg 16) \oplus (d'' \ggg 8) && (5) \\
m_{\sigma_r(2i)} &= c_{\sigma_r(2i+1)} \oplus (a' - b - a) && (6) \\
m_{\sigma_r(2i+1)} &= c_{\sigma_r(2i)} \oplus (a'' - (c'' \oplus (b'' \ggg 25)) - a') && (7)
\end{aligned}$$

First, the intermediate state $v'$ after the column step is computed with the Eqs. (1) and (2) for the $G_i$ functions in the diagonal step, and with the Eqs. (3) and (4) for the $G_i$ functions in the column step. Once $v'$ is fully known, the message words can be computed with (5–7) for all the $G_i$ functions in the $r$-th round.

### *3.1. Collisions for BLOKE-32 with an arbitrary number of rounds*

First we observe that for a fixed state $v$ we can compute the message block $m$, such that $v = \mathit{round}_r(v, m)$, by using the method described above. Since the permutations $\sigma_r$ are all identity in BLOKE, the function $\mathit{round}_r$ does not depend on $r$. Hence, the message block $m$ – let's call it the **fixed-point block** for $v$ – preserves the state $v$ after an arbitrary number of rounds.

We will use a fixed-point block to obtain a collision for BLOKE. Let $h$ be the intermediate hash value before hashing the fixed-point block and $h'$ the intermediate hash value after it. We now express the words $h'_i$ and $h'_{i+4}$ for $i = 0, \ldots, 3$:

$$\begin{aligned} h'_i &= h_i \oplus s_i \oplus h_i \oplus (s_i \oplus c_i) = c_i \\ h'_{i+4} &= h_{i+4} \oplus s_i \oplus h_{i+4} \oplus (t_{i/2} \oplus c_{i+4}) = s_i \oplus t_{i/2} \oplus c_{i+4} \end{aligned}$$

We can see that the intermediate hash value $h'$ only depends on the salt and counter and it does not depend on the previous hash value $h$. Hence we can take any two messages of length $k \cdot 512$ and append a fixed-point block for the intermediate hash value after the original $k$ blocks of the message have been hashed, for each of the two messages. The intermediate hash value after the fixed-points blocks will then be equal for both messages. Since both messages have equal length, the final hash value will be the same even with a padding block.

To demonstrate the collision we produce two messages, each of them two blocks long. The first blocks contain example messages, while the second blocks are fixed-point blocks for the intermediate hash values after the first blocks. The two colliding messages found are

```
5468697320697320616E206578616D706C6520626C6F636B207573656420746F
2070726F647563652061206F6C6C6973696F6E20666F7220424C4F4B453332
0AD8F28C7A8FA271D4110C49592B0A48C5703CF58F07E042E47731724F1349B1
3C80AA3BE14D4017C6DA5E7B93F3E877955ED5B08F078B9C125AB666366AEC0A
```

and

```
616E64207468697320697320616E6F7468657220626C6F636B20757365642074
6F2070726F647563652061206F6C6C6973696F6E20666F7220424C4F4B4521
22BA5209EC48A6BCF0A620BE4581738432F35EDE4CF73121C34D57AB4CB49626
AB700F7678E21C418C01497962A847B239EB418A87694D63B9FE8405269A6DB6
```

The ten-round BLOKE-32 hash of both messages with a padding block and zero salt is

```
F499BEEE73AB52747EDD0EFEE9F739C46E83C33623FDBBF40C25DEE8BD616DFF
```

We can see that using this technique, large multicollision sets can be efficiently computed: we start with an arbitrary set of messages, extend them to the same length $k \cdot 512$, and then append the fixed-point block to each message.

### *3.2. Internal collisions for BRAKE-32 with an arbitrary number of rounds*

Let $h^0$ be an intermediate hash value after $n$ message blocks have been hashed. Let $m^1, m^2, \ldots, m^k$ be the following $k$ message blocks and $h^1, h^2, \ldots, h^k$ the intermediate hash values after each of the message blocks has been hashed. Since no feedforward is used in BRAKE, each word of $h^i$ is computed as $h^i_j = v^i_j \oplus v^i_{j+8}$, where $v^i$ is the state after the chosen number of rounds have been performed for the message block $m^i$. If $m^i$ for $i = 1, \ldots, k$ are all fixed-point blocks for $v^i$, then we can express $v^i$ as

$$v^i = \begin{pmatrix} h^{i-1}_0 & h^{i-1}_1 & h^{i-1}_2 & h^{i-1}_3 \\ h^{i-1}_4 & h^{i-1}_5 & h^{i-1}_6 & h^{i-1}_7 \\ s_0 \oplus c_0 & s_1 \oplus c_1 & s_2 \oplus c_2 & s_3 \oplus c_3 \\ t^i_0 \oplus c_4 & t^i_0 \oplus c_5 & t^i_1 \oplus c_6 & t^i_1 \oplus c_7 \end{pmatrix},$$

where $t^i_0 t^i_1$ is a 64-bit big-endian representation of $(n+i) \cdot 512$. If the message length is smaller than $2^{32}$ bits, then $t^i_1 = 0$ for all $i$ and $t^i_0 = (n+i) \cdot 512$. We will now express the words $h^i_0$ and $h^i_4$ of the intermediate hash values. The expressions for other words are similar.

$$\begin{array}{rcl@{\qquad}rcl} h^1_0 & = & h^0_0 \oplus s_0 \oplus c_0 & h^1_4 & = & h^0_4 \oplus (n+1) \cdot 512 \oplus c_4 \\ h^2_0 & = & h^0_0 & h^2_4 & = & h^0_4 \oplus ((n+1) \oplus (n+2)) \cdot 512 \\ & \cdots & & & \cdots & \\ h^{2i-1}_0 & = & h^0_0 \oplus s_0 \oplus c_0 & h^{2i-1}_4 & = & h^0_4 \oplus (\bigoplus_{j=1}^{2i-1} (n+j)) \cdot 512 \oplus c_4 \\ h^{2i}_0 & = & h^0_0 & h^{2i}_4 & = & h^0_4 \oplus (\bigoplus_{j=1}^{2i} (n+j)) \cdot 512 \end{array}$$

We want to find a pair $(n, k)$ of integers such that $h^k = h^0$. If $k$ is even, the only condition to satisfy is $\bigoplus_{j=1}^{k} (n+j) = 0$. Taking a look at the binary representations of four consecutive numbers, we observe that their two least significant bits are 00, 01, 10 and 11, which XOR together to 00. The remaining bits change at most once – when the two least significant bits change from 11 to 00. Hence, if $n$ is an odd number, then $(n+1) \oplus (n+2) \oplus (n+3) \oplus (n+4) = 0$. By consequence, any sequence of $4r$ consecutive numbers starting with an even number will XOR to 0.

Now we can derive an internal collision attack for BRAKE. We choose an arbitrary message with an odd block length, and extend it by $4r$ fixed-point blocks for their initial state. The intermediate hash value before the padding block will be the same for both messages. Note that the padded messages would have different hash values, as the two messages have different lengths.

Again we produce two messages to demonstrate the collisions. The first message here is an example one-block message, while the second one is its extension by four fixed-point blocks. The two colliding messages found are

```
5468697320697320616E206578616D706C6520626C6F636B207573656420746F
2070726F64756365206120636F6C6C6973696F6E20666F72204252414B453332
```

and

```
5468697320697320616E206578616D706C6520626C6F636B207573656420746F
2070726F64756365206120636F6C6C6973696F6E20666F72204252414B453332
D04D56B8295BC1349F6A576F3D77EF8E5655A4E96F82143781BEE53A4FD47772
D832A666A6E9D1234FD3C42ACE387D1F550FB1E15DAC3848147AAD5A96BE5499
8D3B34DBC85A63898709C18749C10583243491CDCDBABFDD017BF830B95B78AA
413E0F937A68FACFB501B9657D27D033DC1B1968EBA0C92A287A24E1E5D475AF
524D20F82A1E72AC1D6A5D6FBDF56986565644E9CF77778881BFCB7AE971B18E
D6C850AB1594A51E9192F81E8DFBFF2BF51130945DAAB2440AAEA103F6BF5494
C33AF29D0D9369DFA909C38A275123B8245531CDED801C5A016919F025FC4701
754D2D4ADB1A16D8F5ACD5713CEC5627BC1E76DFEB9D5B366B1E28CF85CF75AA
```

The ten-round BRAKE-32 intermediate hash value of both messages with zero salt before the padding block is

```
9EFF3121C2F2256C9D02DE38A65111513AAB290709B43AA3356EF46262F19E37
```

Finally, we note that both attacks also work for the 64-bit versions of BLOKE and BRAKE, with the difference that the Eqs. (1)–(7) are changed to reflect the changes of the rotations in the $G_i$ function.

## 4. Conclusion

We have presented a very efficient method for producing an arbitrary number of collisions for full-round BLOKE, a weakened version of BLAKE in which the message words and constants are not permuted in each round of the compression function, as well as an internal collision attack on the further weakened version BRAKE. The presented attacks thus confirm the necessity of permuting the order in which the message words are used in each round of the compression function of BLAKE, as well as the already established practice for hash functions to include the message length in the padding.

[1] The SHA-3 Zoo. http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo, 2009.

[2] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan. SHA-3 proposal BLAKE. Submission to NIST. http://131002.net/blake/blake.pdf, 2008.

[3] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan. Toy versions of BLAKE. http://131002.net/blake/toyblake.pdf, 2008.

[4] J.-P. Aumasson, W. Meier, and R. C-W. Phan. The hash function family LAKE. In *Fast Software Encryption*, volume LNCS 5086, pages 36–53. Springer-Verlag, 2008.

[5] J.-P. Aumasson, J. Guo, S. Knellwolf, K. Matusiewicz, and W. Meier. Differential and invertibility properties of BLAKE (full version). Cryptology ePrint Archive, Report 2010/043. `http://eprint.iacr.org/2010/043.pdf`, 2010.

[6] D. J. Bernstein. ChaCha, a variant of Salsa20. `http://cr.yp.to/chacha/chacha-20080128.pdf`, 2008.

[7] E. Biham and O. Dunkelman. A framework for iterative hash functions - HAIFA. Second NIST Cryptographic Hash Workshop, 2006. Available at `http://eprint.iacr.org/2007/278.pdf`.

[8] J. Guo and K. Matusiewicz. Round-reduced near-collisions of BLAKE-32. Presented at WEWoRC 2009. `http://www.jguo.org/docs/blake-col.pdf`, 2009.

[9] L. Ji and X. Liangyu. Attacks on round-reduced BLAKE. Cryptology ePrint Archive, Report 2009/238. `http://eprint.iacr.org/2009/238.pdf`, 2009.

[10] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, 1997.

[11] National Institute of Standards and Technology. Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family. Federal Register, vol. 72(212). `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`, 2007.

[12] L. Wang, K. Ohta, and K. Sakiyama. Free-start preimages of step-reduced Blake compression function. Rump session of ASIACRYPT 2009. `http://asiacrypt2009.cipher.risk.tsukuba.ac.jp/rump/slides/11.pdf`, 2009.