Pseudo-cryptanalysis of the Original Blue Midnight Wish

Søren S. Thomsen
DTU Mathematics, Technical University of Denmark

September 28, 2009

Abstract

The hash function *Blue Midnight Wish* (BMW) is a candidate in the SHA-3 competition organised by the U.S. National Institute of Standards and Technology (NIST). BMW was selected for the second round of the competition, but the algorithm was tweaked in a number of ways. In this paper we describe cryptanalysis on the original version of BMW, as submitted to the SHA-3 competition in October 2008. When we refer to BMW, we therefore mean the original version of the algorithm.

The attacks described are (near-)collision, preimage and second preimage attacks on the BMW compression function. These attacks can also be described as pseudo-attacks on the full hash function, i.e., as attacks in which the adversary is allowed to choose the initial value of the hash function. The complexities of the attacks are about 2^{14} for the near-collision attack, about $2^{3n/8+1}$ for the pseudo-collision attack, and about $2^{3n/4+1}$ for the pseudo-(second) preimage attack, where n is the output length of the hash function. Memory requirements are negligible. Moreover, the attacks are not (or only moderately) affected by the choice of security parameter for BMW.

Keywords: hash function cryptanalysis, SHA-3 competition, Blue Midnight Wish, pseudo-attacks.

1 Introduction

On October 31, 2008, the "SHA-3 competition", organised by the National Institute of Standards and Technology (NIST), was launched [5]. 64 algorithms were submitted, and 51 of these were accepted for the first round of the competition. On July 24, 2009, 14 candidates were chosen by NIST to advance to the second round of the competition.

One of the candidates accepted for the second round is called *Blue Midnight Wish* [3], or BMW for short, and was developed by Gligoroski, Klíma, Knapskog, El-Hadedy, Amundsen, and Mjølsnes. On a standard PC, BMW is arguably the most efficient of the 14 hash functions remaining in the competition [1]. Blue Midnight Wish was tweaked for the second round of the competition. In this paper, we **always** refer to the original version of the hash function, i.e., the version submitted to the competition on (or before) October 31, 2008.

In this paper we describe some weaknesses in Blue Midnight Wish. We show how to easily find near-collisions in the compression function of Blue Midnight Wish. By near-collisions we mean a pair of inputs to the compression function for which the outputs only differ in a few (pre-specified) bit positions.

We also show how to find collisions, preimages, and second preimages in the compression function, faster than what is possible for an ideal compression function. This can be done by controlling 128, respectively 256 bits of the output of the compression function of BMW-256, respectively BMW-512. By controlling we mean that the adversary can give these bits any value he wishes with negligible effort. These attacks on the compression function can be extended to become pseudo-attacks on the full hash function, i.e., attacks which allow the adversary to pick the initial value of the hash function. The complexity of these attacks corresponds to the complexity of a 192-bit, respectively 384-bit ideal hash function in the case of BMW-256, respectively BMW-512. Hence, for instance, pseudo-collisions can be found in BMW-512 in time about 2^{192}, which is to be compared to the expected 2^{256} for an ideal 512-bit hash function. Memory requirements of all attacks are negligible.

We point out that the attacks described in this paper do not seem to apply to the tweaked version of Blue Midnight Wish. We are making these results public in the hope that they will somehow be useful to other cryptanalysts.

1.1 Organisation of the Paper

We explain the notation used throughout the paper in the next subsection. Then, in Section 2 we describe how the Blue Midnight Wish hash functions work. In Section 3 we describe the near-collision attack on the compression function. In Section 4, we describe a more general pseudo-attack, which may be used in pseudo-collision, -preimage, and -second preimage attacks, and we give some concrete examples. In Section 5 we conclude.

1.2 Notation

The symbol '$\oplus$' denotes exclusive-or (XOR). $x^{\ll s}$ means x shifted left bitwise by s positions; similarly, $x^{\gg s}$ means x shifted right by s positions. By $x^{\lll s}$ we mean x rotated (cyclically shifted) left by s positions. The expression $x\|y$ denotes the concatenation of the two bit strings x and y, and we denote the bitlength of x by $|x|$.

2 A Description of Blue Midnight Wish

Blue Midnight Wish is in fact a collection of four hash functions returning digests of four different sizes: 224 bits, 256 bits, 384 bits, and 512 bits. The two shortest digests are computed in the same way, except in the final step, which is a truncation. Likewise for the two longest digests. The word size, denoted by w, for the short variants is 32 bits, and for the long variants is 64 bits. Apart from the word size, all four variants are very similar. A little-endian byte ordering is assumed.

Blue Midnight Wish applies only four different types of operations: additions modulo 2^w, exclusive-ors (XORs), and bitwise shifts and rotations. In the following, all additions of words in the description of Blue Midnight Wish are to be taken modulo 2^w.

Blue Midnight Wish maintains a state of 16 words during the processing of a message; only in the end, the 16 words are truncated down to 6, 7, or 8, depending on the digest size (truncation is done by throwing away the first 10, 9, or 8 words, respectively). Message blocks are also 16 words in length, and Blue Midnight Wish operates with a compression function mapping 2×16 words to 16 words. The compression function is iterated in a standard fashion [2, 4]. Hence, the message m of bitlength $\mu = |m|$ must be padded to a length that is a multiple of $16w$ bits, which is done by first appending a

'1'-bit, then appending $z = -\mu - 65 \bmod 16w$ '0'-bits (this part of the padding will be called "10..." padding in the following), and finally appending a 64-bit representation of the message length μ (length padding).

We now turn to a description of the Blue Midnight Wish compression function.

2.1 The Blue Midnight Wish Compression Function

The Blue Midnight Wish compression function takes two 16-word inputs and returns a single 16-word output. It applies three different sub-functions, which are called P, f_1, and f_2. P is an efficiently invertible permutation[1]. f_1 is a so-called multi-permutation taking two inputs, meaning that by fixing one of the inputs, the function is a permutation on the other input. Finally, f_2 compresses three inputs of 16 words to a single 16-word output, which is also the output of the compression function.

The two 16-word inputs to the compression function will be called H and M, H being the *chaining variable*, and M being the *message block*. Referring to a single word in one of the inputs will be denoted by H_i or M_i, meaning word number i, where counting starts from 0. Hence, (e.g.) $M = M_0 \| M_1 \| \cdots \| M_{15}$.

The input to the permutation P is $H \oplus M$. The output of P is denoted $Y = Y_0 \| Y_1 \| \cdots \| Y_{15}$. The inputs to f_1 are Y and M. The output of f_1 is denoted $Z = Z_0 \| Z_1 \| \cdots \| Z_{15}$. The inputs to f_2 are Y, Z, and M, and the output, which is also the output of the compression function, is denoted H^*. See also Figure 1.

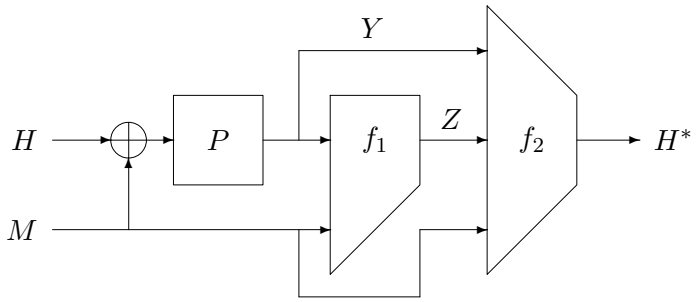

Figure 1: The Blue Midnight Wish compression function.

Given Y and M, H can easily be computed as $P^{-1}(Y) \oplus M$, since P is efficiently invertible. Moreover, H is not used as input to any other sub-function than P. Hence, in attacks on the compression function, the details of P are irrelevant, and therefore we postpone a description of these to Appendix A. We now describe the two other components of the compression function.

2.1.1 A Description of f_1.

As mentioned, the inputs to f_1 are M and Y, and the output is denoted Z. Let $Q = Y \| Z$ be a 32-word vector, and note that when f_1 is called, Q contains 16 already computed words, and 16 "null" words. Then, f_1 can be described as a shift register, that computes one word of Q at the time as a function of the previous 16 words of Q.

There are two variants of the step function that computes each new word of Q: a simple step function, and a more complex one. Since 16 words are computed in f_1,

[1] In the Blue Midnight Wish specification, a mapping $f_0 : \{0,1\}^{16w} \times \{0,1\}^{16w} \rightarrow \{0,1\}^{16w}$ is defined. Since f_0 is a permutation on the XOR of its two inputs, we choose to focus on this permutation and denote it P, i.e., $f_0(H, M) = P(H \oplus M)$.

there are always 16 rounds, but the number of complex and simple rounds depends on a tunable security parameter. By default, there are first two complex rounds and then 14 simple rounds, and we shall generally assume that this is the distribution of complex and simple rounds. However, all our attacks apply to BMW using any value of the security parameter (in the case of the near-collision attack, some modifications are required).

Both complex and simple rounds use a number of invertible sub-functions $s_0, \ldots, s_5$ and $r_1, \ldots, r_7$, whose descriptions are postponed to Appendix B. Both types of rounds also use the same "message schedule": consider M to be a 16-element column vector in $\mathbb{Z}_{2^w}$, and define the matrix $\mathbf{B}$ as the circulant matrix whose first row contains the 16 elements

$$[1, 0, 0, 1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0].$$

Row $i + 1$ of $\mathbf{B}$ is row i cyclically shifted one position to the right. Note that $\mathbf{B}$ is invertible for both word sizes 32 and 64; the inverses can be found in Appendix C. Define $W = \mathbf{B} \cdot M \bmod 2^w$, with W_i referring to the ith word of W. In round i of f_1, W_i is involved in the computation of $Z_i = Q_{i+16}$.

Furthermore, 16 constants K_i are defined as $\lfloor 2^w/3 \rfloor \cdot i$. We can now describe a complex round as

$$\begin{aligned} Q_{i+16} \leftarrow\ & s_1(Q_i) + s_2(Q_{i+1}) + s_3(Q_{i+2}) + s_0(Q_{i+3}) \\ & + s_1(Q_{i+4}) + s_2(Q_{i+5}) + s_3(Q_{i+6}) + s_0(Q_{i+7}) \\ & + s_1(Q_{i+8}) + s_2(Q_{i+9}) + s_3(Q_{i+10}) + s_0(Q_{i+11}) \\ & + s_1(Q_{i+12}) + s_2(Q_{i+13}) + s_3(Q_{i+14}) + s_0(Q_{i+15}) \\ & + W_i + K_i, \end{aligned}$$

for increasing i; for the default choice of the tunable security parameter, i increases from 0 to 1. A simple round, covering the range of remaining values of i up to and including 15, is described as

$$\begin{aligned} Q_{i+16} \leftarrow\ & Q_i + r_1(Q_{i+1}) + Q_{i+2} + r_2(Q_{i+3}) \\ & + Q_{i+4} + r_3(Q_{i+5}) + Q_{i+6} + r_4(Q_{i+7}) \\ & + Q_{i+8} + r_5(Q_{i+9}) + Q_{i+10} + r_6(Q_{i+11}) \\ & + Q_{i+12} + r_7(Q_{i+13}) + s_5(Q_{i+14}) + s_4(Q_{i+15}) \\ & + W_i + K_i. \end{aligned}$$

Note that given M and 16 consecutive words of Q, the remaining 16 words of Q can be computed; in particular, given M and Z, Y can be computed. Likewise, given Y and Z (i.e., all of Q), M can be computed via W as $M = \mathbf{B}^{-1} \cdot W \bmod 2^w$.

2.1.2 A Description of f_2.

The sub-function f_2 takes as input M, Y, and Z. Let

$$\begin{aligned} X_L &= Z_0 \oplus Z_1 \oplus \cdots \oplus Z_7 \quad \text{and} \\ X_H &= Z_0 \oplus Z_1 \oplus \cdots \oplus Z_{15}. \end{aligned}$$

The output words $H^*_0, \ldots, H^*_{15}$ are computed as follows.

$$H^*_0 = (X_H^{\lll 5} \oplus Z_0^{\ggg 5} \oplus M_0) + (X_L \oplus Z_8 \oplus Y_0)$$

$$
\begin{aligned}
H_1^* &= (X_H^{\gg 7} \oplus Z_1{}^{\ll 8} \oplus M_1) + (X_L \oplus Z_9 \oplus Y_1) \\
H_2^* &= (X_H^{\gg 5} \oplus Z_2{}^{\ll 5} \oplus M_2) + (X_L \oplus Z_{10} \oplus Y_2) \\
H_3^* &= (X_H^{\gg 1} \oplus Z_3{}^{\ll 5} \oplus M_3) + (X_L \oplus Z_{11} \oplus Y_3) \\
H_4^* &= (X_H^{\gg 3} \oplus Z_4 \oplus M_4) + (X_L \oplus Z_{12} \oplus Y_4) \\
H_5^* &= (X_H^{\ll 6} \oplus Z_5{}^{\gg 6} \oplus M_5) + (X_L \oplus Z_{13} \oplus Y_5) \\
H_6^* &= (X_H^{\gg 4} \oplus Z_6{}^{\ll 6} \oplus M_6) + (X_L \oplus Z_{14} \oplus Y_6) \\
H_7^* &= (X_H^{\gg 11} \oplus Z_7{}^{\ll 2} \oplus M_7) + (X_L \oplus Z_{15} \oplus Y_7) \\
H_8^* &= (H_4^*)^{\lll 9} + (X_H \oplus Z_8 \oplus M_8) + (X_L^{\ll 8} \oplus Z_7 \oplus Y_8) \\
H_9^* &= (H_5^*)^{\lll 10} + (X_H \oplus Z_9 \oplus M_9) + (X_L^{\gg 6} \oplus Z_0 \oplus Y_9) \\
H_{10}^* &= (H_6^*)^{\lll 11} + (X_H \oplus Z_{10} \oplus M_{10}) + (X_L^{\ll 6} \oplus Z_1 \oplus Y_{10}) \\
H_{11}^* &= (H_7^*)^{\lll 12} + (X_H \oplus Z_{11} \oplus M_{11}) + (X_L^{\ll 4} \oplus Z_2 \oplus Y_{11}) \\
H_{12}^* &= (H_0^*)^{\lll 13} + (X_H \oplus Z_{12} \oplus M_{12}) + (X_L^{\gg 3} \oplus Z_3 \oplus Y_{12}) \\
H_{13}^* &= (H_1^*)^{\lll 14} + (X_H \oplus Z_{13} \oplus M_{13}) + (X_L^{\gg 4} \oplus Z_4 \oplus Y_{13}) \\
H_{14}^* &= (H_2^*)^{\lll 15} + (X_H \oplus Z_{14} \oplus M_{14}) + (X_L^{\gg 7} \oplus Z_5 \oplus Y_{14}) \\
H_{15}^* &= (H_3^*)^{\lll 16} + (X_H \oplus Z_{15} \oplus M_{15}) + (X_L^{\gg 2} \oplus Z_6 \oplus Y_{15}).
\end{aligned}
$$

3 Near-collisions in the Compression Function

Attacks on the compression function of Blue Midnight Wish are not affected by the permutation P, since this permutation can be inverted, and thereby the chaining input can be computed.

One may also observe that by choosing the same (XOR) differences in H and M, there is no input difference in P, and therefore also no output difference. By ensuring that only the last few words of the expanded message W contain a difference, we see that no difference is involved in a large part of f_1. Combined with the fact that diffusion is not very effective in f_2, this observation leads to near-collisions in the compression function.

Hence, a strategy to find the best (lowest weight) near-collision in the compression function is to search for difference patterns of the last few words of W, such that these differences do not spread too much in the last few rounds of f_1 and in f_2. Note that the inverse message schedule must be applied to W in order to be able to compute f_2, and this message schedule will cause some diffusion of differences in W; however, differences in the most significant bits (MSBs) will remain in the MSB positions after the inverse message schedule is applied. Therefore, an obvious choice is to search for difference patterns in W that only affect the MSBs of words of W.

3.1 An Example

In the case of both BMW-256 and BMW-512, the search mentioned showed that a good difference pattern in W has differences in the MSBs of W_{13}, W_{14}, and W_{15} only. The inverse message schedule causes differences in the MSBs of M_0, M_1, M_3, M_4, M_7, M_9, and M_{13}. Hence, there are 7 bit differences in M, which are introduced in the function f_2.

Table 2: The desired binary differences in the last three words of Z.

Word	Desired XOR difference (binary)
Z_{13}	100...0
Z_{14}	010...0
Z_{15}	110...0

A difference in W_{13} is propagated directly to Z_{13} in the 13th round of f_1. Hence, Z_{13} obtains the difference $100\ldots0$ (in binary). In round 14, the function s_4 is applied to Z_{13} yielding the difference $1100\ldots0$ (see Appendix B), and a difference in the MSB of W_{14} is also introduced. The end result is the difference $0100\ldots0$ in Z_{14} with probability 1/2. Finally, in round 15, the function s_5 is applied to Z_{13} yielding the difference $10100\ldots0$, the function s_4 is applied to Z_{14} yielding the difference $01100\ldots0$, and a difference in the MSB of W_{15} is also introduced. Optimally, these differences result in the difference $1100\ldots0$ in Z_{15}, since then $\Delta Z_{13} \oplus \Delta Z_{14} \oplus \Delta Z_{15} = 0$ (ΔZ_i meaning the difference on Z_i), which means that in f_2, the variables X_L and X_H will contain no difference. The total probability of this characteristic is about 2^{-3}. See Table 2.

In f_2, as mentioned, the desired bit differences in Z yield no difference in X_L and X_H. Hence, in the output words $H_0^*, H_1^*, H_3^*, H_4^*$, there are only differences in the MSBs, and these come from the message M. There is no difference in H_2^*. In H_5^*, the MSB difference in Z_{13} is inherited, and there are no other differences. In H_6^*, the difference $0100\ldots0$ in Z_{14} is inherited and with probability 1/2 does not propagate. In H_7^*, the MSB difference in M_7 cancels the MSB difference in Z_{15}, and the resulting difference is $0100\ldots0$, which does not propagate with probability 1/2.

In H_8^*, the difference coming from H_4^* is rotated and does not propagate with probability 1/2. When it does not propagate, the difference is $0\ldots0100000000$. In H_9^*, differences come from H_5^* and M_9. The MSB difference in H_5^* is rotated by 10 positions to yield $0\ldots01000000000$. With probability 1/2, it does not propagate. The MSB difference from M_9 is inherited. In H_{10}^*, the difference from H_6^* is rotated by 11 positions to yield $0\ldots01000000000$ with probability 1/2. In H_{11}^*, the difference from H_7^* is rotated by 12 positions to yield $0\ldots010000000000$ with probability 1/2. In H_{12}^*, the MSB difference from H_0^* is rotated by 13 positions to yield $0\ldots01000000000000$ with probability 1/2. In H_{13}^*, there are differences in three input words: H_1^*, Z_{13}, and M_{13}. However, the MSB differences in Z_{13} and M_{13} cancel each other out, and the MSB difference in H_1^* yields $0\ldots010000000000000$ with probability 1/2. In H_{14}, the difference $0100\ldots0$ from Z_{14} is inherited with probability 1/2, and this is the only input difference. Finally, in H_{15}^* there are differences coming from H_3^* and Z_{15}; these yield $0100\ldots01000000000000000$ with total probability 1/4.

To sum up, there are one-bit differences in $H_0^*, H_1^*, H_3^*, \ldots, H_8^*, H_{10}^*, \ldots, H_{14}^*$, and two-bit differences in H_9^* and H_{15}^* (see Table 3). The total number of bit differences is 17, and the total probability of this near-collision is about 2^{-14} (assuming independence). 10 of the differences are in the words $H_8^*, \ldots, H_{15}^*$, which are the output words of the BMW-256 and BMW-512 hash functions (after truncation). 9 differences are in the words $H_9^*, \ldots, H_{15}^*$, which are the output words of the BMW-224 hash function. 7 differences are in the words $H_{10}^*, \ldots, H_{15}^*$, which are the output words of the BMW-384 hash function.

Table 3: Output differences in the near-collision attack on the BMW compression function. Apply to all variants.

Word	XOR difference (binary)
H_0^*	100...00000000000000000
H_1^*	100...00000000000000000
H_2^*	000...00000000000000000
H_3^*	100...00000000000000000
H_4^*	100...00000000000000000
H_5^*	100...00000000000000000
H_6^*	010...00000000000000000
H_7^*	010...00000000000000000
H_8^*	000...00000000100000000
H_9^*	100...00000001000000000
H_{10}^*	000...00000001000000000
H_{11}^*	000...00000010000000000
H_{12}^*	000...00001000000000000
H_{13}^*	000...00010000000000000
H_{14}^*	010...00000000000000000
H_{15}^*	010...01000000000000000

3.2 Other Difference Patterns

We note that the difference in Z may be slightly different, and still give the same results as those described. For instance, the difference patterns of Z_{14} and Z_{15} may be swapped.

Moreover, there are in fact slightly better message difference patterns than the one described above. As an example, a difference in the MSB of W_{13} and in the second-most significant bit of W_{15} yields—with a high probability—a near-collision in all but 14 bits of the compression function output (9 of these are in the last eight words). However, the corresponding message difference in M has a higher Hamming weight, and specifically there are differences in the words M_{14} and M_{15}, which (in a pseudo-attack on the hash function) are reserved for padding. We did not find simple difference patterns with differences only in the last few words of W that lead to full collisions with high probability.

With a value of the security parameter above 13, the above characteristic has a low (if not zero) probability. However, even with a value of 16, a high probability characteristic exists producing near-collisions of total Hamming weight as low as 24 for the 16 output words of the compression function (see [6]).

3.3 A Pseudo-near-collision in BMW-256

In the attack described above, there are no differences in M_{14} and M_{15}, which in BMW-256 are the words containing length padding. This means we can extend the near-collision attack on the compression function to a pseudo-near-collision attack on the BMW-256 hash function. Moreover, one of the colliding messages may start from the correct initial value of BMW-256; the other initial value will be different from the correct one in the same 7 bit positions as those which contain differences in M.

As an example, the bit sequence of length 447 bits starting with the three bytes `f3 8b 01` and ending with (423) '0'-bits follows the characteristic described above (with chaining input equal to the BMW-256 initial value). The 256-bit output of the hash function is

`7927dc5d5759cdd85bdd3e8276e41dd5fd627bff0f3813f51238db090a5e433d`.

When applying the difference pattern described above to both the message and the initial value of BMW-256, the hash function output is

`7926dc5d575bcd585bdf3e8276e01dd5fd727bff0f1813f51238db490ade437d`.

The XOR of the two outputs is

`0001000000020080000200000004000000100000002000000000004000800040`.

4 Pseudo-attacks

A second observation on the BMW compression function leads to improved pseudo-collision, -preimage, and -second preimage attacks: if $Z_i = 0$ for all i, $0 \leq i < 16$, then we get the following greatly simplified description of f_2.

$$\begin{aligned} H_0^* &= M_0 + Y_0 \\ H_1^* &= M_1 + Y_1 \end{aligned}$$

$$
\begin{aligned}
H_2^* &= M_2 + Y_2 \\
H_3^* &= M_3 + Y_3 \\
H_4^* &= M_4 + Y_4 \\
H_5^* &= M_5 + Y_5 \\
H_6^* &= M_6 + Y_6 \\
H_7^* &= M_7 + Y_7 \\
H_8^* &= (M_4 + Y_4)^{\lll 9} + M_8 + Y_8 \\
H_9^* &= (M_5 + Y_5)^{\lll 10} + M_9 + Y_9 \\
H_{10}^* &= (M_6 + Y_6)^{\lll 11} + M_{10} + Y_{10} \\
H_{11}^* &= (M_7 + Y_7)^{\lll 12} + M_{11} + Y_{11} \\
H_{12}^* &= (M_0 + Y_0)^{\lll 13} + M_{12} + Y_{12} \\
H_{13}^* &= (M_1 + Y_1)^{\lll 14} + M_{13} + Y_{13} \\
H_{14}^* &= (M_2 + Y_2)^{\lll 15} + M_{14} + Y_{14} \\
H_{15}^* &= (M_3 + Y_3)^{\lll 16} + M_{15} + Y_{15}.
\end{aligned}
$$

4.1 Controlling Output Words – A First Example

Some output words of the compression function can be controlled by an attacker after fixing $Z = 0$. The idea is to fix some words of M and some words of Y in such a way that a number of output words obtain an arbitrary value chosen by the attacker, and such that f_1 can be computed backwards, i.e., computing Y_{15} from Z, then Y_{14}, etc. Words of M can be fixed directly, since they are part of the input to the compression function. Words of Y can be fixed indirectly via words of W, which, as mentioned, depend on M. There is enough freedom to fix some words of M and some words of W at the same time. More details follow. Note that this attack is independent of the value of the security parameter, since both simple and complex rounds are invertible.

Consider as an example the "new" definition of H_{11}^*:

$$H_{11}^* = (M_7 + Y_7)^{\lll 12} + M_{11} + Y_{11}.$$

By fixing M_7, Y_7, M_{11}, and Y_{11}, one has effectively controlled H_{11}^*. Message words are part of the input to the compression function. Words of Y can be controlled to some extent via words of W; after having fixed Z, we are able to compute words of Y in the backward direction, i.e., we compute Y_{15} first, then Y_{14} etc., all the way down to Y_0. Alternatively, we can compute the value of W_i needed to get some desired value of Y_i, for any i such that Y_j is already computed for all $j > i$. Thereby, we indirectly control Y_i.

As a simple example for BMW-256, assume we want H_{11}^* to obtain the value α. To do this, we may choose (e.g.) $M_7 = Y_7 = Y_{11} = 0$ and $M_{11} = \alpha$. We obtain $Y_7 = Y_{11} = 0$ by controlling W_7 and W_{11}. Note that once Y and M are fixed, we compute H as described in Section 2.1. How to control words of M and words of W at the same time is now described.

Sticking to the example, assume we want to be able to control M_7, M_{11}, W_7, and W_{11}. Compute $W = \mathbf{B} \cdot M$ with (initially) all words of M as free parameters. As an example, one gets $W_{15} = M_2 - M_9 + M_{15}$. Now, make W_{15} free by replacing everywhere M_{15} by $W_{15} - M_2 + M_9$. Now W_{15} is freed, but M_{15} is no longer free.

Since $W_{14} = M_1 - M_8 + M_{14}$, we can make W_{14} free by replacing everywhere M_{14} by $W_{14} - M_1 + M_8$. We may continue like this, freeing all W_i down to $i = 7$ (incl.),

without making M_7 or M_{11} dependent. Since M_{13}, M_{14}, and M_{15} contain padding, we might want to keep these three words of M free as well. This way, one obtains (e.g.) the following value of W (where M_1, M_3, M_7, M_{11}, M_{13}, M_{14}, and M_{15} are free words of M, and $W_7, \ldots, W_{15}$ are free words of W; all other words of M and W are dependent):

$$\begin{bmatrix} -M_1 + M_3 + 2M_7 - M_{13} - W_7 + W_{13} \\ 2M_1 - M_7 - M_{11} + M_{13} + W_7 - W_{10} \\ 2M_1 - M_3 + 3M_{11} + 3M_{14} + M_{15} - 2W_8 - W_9 - W_{11} - 2W_{14} - W_{15} \\ -M_1 + 2M_3 - M_{11} - M_{13} - M_{14} + W_8 + W_9 - W_{12} + W_{14} + W_{15} \\ M_1 + M_{13} - M_{14} + W_7 - W_{10} \\ M_1 + M_{11} + 2M_{14} - M_{15} - W_{11} - W_{14} \\ M_3 - M_7 + M_{13} + M_{15} + W_9 - W_{12} - W_{13} \\ W_7 \\ \vdots \\ W_{15} \end{bmatrix}.$$

By computing the words Y_i in the backward direction, or choosing Y_i and computing the required W_i for i from 15 down to 7, we can control all the words $Y_{15}, Y_{14}, \ldots, Y_7$. In particular, we can make sure that $Y_7 = Y_{11} = 0$. Since M_7 and M_{11} are free, we can also choose these two message words as we want; in particular, we can choose $M_7 = 0$ and $M_{11} = \alpha$, so that we obtain $H^*_{11} = \alpha$. Since we indirectly also control H^*_7, we can obtain $H^*_7 = \beta$ for any β of our choice via a proper choice of, say, M_7.

Note that in order to compute Y_1 and Y_0, s_1 must be inverted (see Section 2.1.1). This is *slightly* more complicated in practise than computing s_1 in the forward direction, but it can also be done efficiently (with some additional memory requirements) by precomputing and storing all inverses.

The reason for choosing to control H^*_{11} is that Y_7 is involved in its computation. This means we have to make only a few words of W free (W_{15} down to W_7), and there is still a large amount of freedom in the choice of a number of words of M. This will be useful in extensions of the attack.

4.2 Controlling Additional Output Words

There are many degrees of freedom left. These can be used to control additional output words. For instance, we may control H^*_6 and H^*_{10} via M_6, W_6, M_{10}, and W_{10}. We again keep M_{14} and M_{15} free as above, but M_{13} is not free. We shall obtain correct "10..." padding in M_{13} probabilistically; the probability is about 1/2 if we assume only a single bit of "10..." padding (hence, the message length is $512 - 65 = 447$ bits). We set $M_6 = M_7 = M_{10} = M_{11} = 0$ (for the sake of simplicity), and now we want to free all W_i for i from 15 down to 6, since we need to be able to control Y_6. Using the same method

as in the previous examples, we obtain the vector

$$\begin{bmatrix} 2M_{14} + M_{15} - W_6 - 2W_7 - 2W_8 - W_9 + W_{12} - 2W_{14} - 2W_{15} \\ -M_{14} - M_{15} + W_6 + W_8 - W_{10} + W_{13} + W_{14} + W_{15} \\ 2M_{14} + M_{15} - W_7 - W_8 - W_{11} - W_{12} - W_{14} \\ 2M_{14} + M_{15} - W_6 - 2W_7 - 2W_8 - W_9 + W_{12} - W_{13} - 2W_{14} - 2W_{15} \\ -2M_{14} - M_{15} + W_6 + W_7 + W_8 - W_{10} + W_{13} + W_{14} + W_{15} \\ 2M_{14} - M_{15} - W_7 - W_{11} - W_{14} \\ W_6 \\ \vdots \\ W_{15} \end{bmatrix}.$$

We now control the four output words H_6^*, H_7^*, H_{10}^*, and H_{11}^* via W_6, W_7, W_{10}, and W_{11}. The time complexity of this attack is about 2, since we need correct "10..." padding in M_{13}, but we have no (direct) control over this message word.

4.3 Other Variants of BMW

The same technique as described above for BMW-256 can be applied to BMW-512. In fact, for BMW-512, length padding takes up only one message word, and therefore we have enough freedom to ensure correct "10..." padding with probability 1.

Obviously, the attacks also apply to BMW-224 and BMW-384, since these differ from BMW-256 and BMW-512 (respectively) only in the initial value and the final truncation.

4.4 Applications

After truncation, two out of eight (or out of six or seven in the case of BMW-384 and BMW-224, respectively) output words can be given any value chosen by the attacker. This control can be used to carry out pseudo-attacks, i.e., attacks in which the attacker is free to choose the initial value of the hash function. Example pseudo-attacks are pseudo-collision, pseudo-preimage, and pseudo-second preimage attacks. The time complexities of these attacks on BMW correspond to brute force attacks on 3/4 of the output bits (or 2/3 or 5/7 in the case of BMW-384 and BMW-224, respectively). Hence, the time complexity is reduced compared to an ideal hash function. Table 5 summarises the attack complexities for the three types of attack on the four variants of Blue Midnight Wish. Memory requirements are negligible.

As mentioned, pseudo-attacks are attacks in which the attacker is free to choose the initial value of the hash function. In the case of pseudo-collision and pseudo-second preimage attacks, the two colliding messages will generally assume two different initial values.

4.5 Available Degrees of Freedom

Clearly, in these attacks we do not have to choose Z to be all-zero, we can choose it to be anything we want. Also, we have lots of freedom in the choice of M_6, M_7, M_{10},

Table 5: Pseudo-attack complexities on the four Blue Midnight Wish variants (in brackets, birthday/brute force complexities).

Variant	Pseudo-collision	Pseudo-(second) preimage
BMW-224	2^{81} (2^{112})	2^{161} (2^{224})
BMW-256	2^{97} (2^{128})	2^{193} (2^{256})
BMW-384	2^{128} (2^{192})	2^{256} (2^{384})
BMW-512	2^{192} (2^{256})	2^{384} (2^{512})

M_{11}, H_6^*, and H_7^* to get the desired values of H_{10}^* and H_{11}^*. The choices we made in the examples above were only to simplify expressions. The available degrees of freedom may be useful in extensions of the attacks; however, so far we did not succeed in doing this.

4.6 Some Examples

As a proof of concept, we now give some examples where output words H_6^*, H_7^*, H_{10}^*, and H_{11}^* are controlled (set to zeros). All words are written in (little endian) hexadecimal.

4.6.1 BMW-256.

With chaining input

$$H = (\texttt{ca9f6a93}, \texttt{3cf76d1d}, \texttt{9e4dc0d0}, \texttt{c3c23fdc}, \texttt{a9dbaa8d}, \texttt{fc21aac3}, \texttt{9b4025d4}, \texttt{9cc22fd7}, \texttt{61d87164}, \texttt{798a37d0}, \texttt{600c7d3d}, \texttt{34e25b56}, \texttt{ed8daa32}, \texttt{6cf3fb69}, \texttt{30fab149}, \texttt{dc9bd4c2})$$

and message input (including padding)

$$M = (\texttt{0803a93a}, \texttt{5013d06d}, \texttt{ebd8f1b6}, \texttt{ada77a2b}, \texttt{41491ede}, \texttt{3af1638e}, \texttt{00000000}, \texttt{00000000}, \texttt{7445bdbc}, \texttt{90eec654}, \texttt{00000000}, \texttt{00000000}, \texttt{610b24e8}, \texttt{11a24bca}, \texttt{000001bf}, \texttt{00000000}),$$

the output of the compression function is

$$H^* = (\texttt{a321e2f8}, \texttt{ec7c4e0a}, \texttt{47f45671}, \texttt{2009f6ed}, \texttt{964a5ce7}, \texttt{5c4fce1d}, \texttt{00000000}, \texttt{00000000}, \texttt{2d7273b0}, \texttt{928f72bc}, \texttt{00000000}, \texttt{00000000}, \texttt{6ca15abe}, \texttt{3fa46015}, \texttt{b16696b4}, \texttt{f6ad9f5c}).$$

Hence, if H as defined above had been the initial value of BMW-256, then the message M would hash to

$$\texttt{b073722dbc728f920000000000000000be5aa16c1560a43fb49666b15c9fadf6}.$$

Finding this chaining/message pair required the equivalent of two compression function calls.

4.6.2 BMW-512.

An example for BMW-512 is the following. With chaining input

$$H = (\texttt{ff95021298b8053b}, \texttt{bcc99f334f6e20af}, \texttt{7387085dbb945b21}, \texttt{a6e3e0c15e1518d3}, \texttt{ac1269c4776c3dce}, \texttt{862544c1ccc9b56e}, \texttt{ba4d394db6052685}, \texttt{fd9cbbf7da10c97c}, \texttt{d5fb04f37d785c9d}, \texttt{5369199dce565dd4}, \texttt{7bdc4c99a89666a9}, \texttt{5c856b804801cf84}, \texttt{a9d3c3347f1e1021}, \texttt{7cb27b3b516d1642}, \texttt{2b80503146c44e2b}, \texttt{130d06d4893028ba})$$

and message input (including padding)

$$\begin{aligned} M \;=\; (&\texttt{1462e93afb35c1cc}, \texttt{da9f0d9ea7f0d9ec}, \texttt{57b861f574387fb3}, \texttt{d8db2240ca1fddca},\\ &\texttt{0b2aee5199e34ed8}, \texttt{00d79aaa54b1b95e}, \texttt{0000000000000000}, \texttt{0000000000000000},\\ &\texttt{2de5cb945577564d}, \texttt{cc627a486df4b73d}, \texttt{0000000000000000}, \texttt{0000000000000000},\\ &\texttt{a4791f4f65cd3bf3}, \texttt{63cfeaaf0067f03c}, \texttt{0100000000000000}, \texttt{00000000000003bf}), \end{aligned}$$

the output of the compression function is

$$\begin{aligned} H^* \;=\; (&\texttt{88e74c537efd26d4}, \texttt{1ac48d4de6bc9df7}, \texttt{dbc7bd6d95876c96}, \texttt{593ce88619782657},\\ &\texttt{ff01649945976d60}, \texttt{6df14428399268cd}, \texttt{0000000000000000}, \texttt{0000000000000000},\\ &\texttt{f63e5fc6672c86f8}, \texttt{100b2869c2be6b8f}, \texttt{0000000000000000}, \texttt{0000000000000000},\\ &\texttt{e63005f1f87f3560}, \texttt{af255aa7c7e0f16c}, \texttt{a7652904d6f61a35}, \texttt{b7dadc75cabe3b7b}). \end{aligned}$$

Hence, if the initial value of BMW-512 would be equal to the chaining variable above, then the 512-bit message digest would be

$$\begin{aligned} &\texttt{f8862c67c65f3ef68f6bbec269280b10}\texttt{0000000000000000000000000000000}\\ &\texttt{60357ff8f10530e66cf1e0c7a75a25af351af6d6042965a77b3bbeca75dcdab7}. \end{aligned}$$

Finding this chaining/message pair required the equivalent of one compression function call.

5 Conclusion

We have described a number of weaknesses in the Blue Midnight Wish hash function.

The weaknesses lead to attacks in which the adversary is allowed to choose the initial value of the hash function. It is by no means straightforward to extend the attacks to full-blown attacks using the given initial values of the BMW variants. Meet-in-the-middle attacks also do not seem possible since BMW uses an internal state that is at least twice as large as the output of the hash function.

The attacks apparently do not apply to the tweaked version of BMW.

References

[1] D. J. Bernstein and T. Lange (editors). eBACS: ECRYPT Benchmarking of Cryptographic Systems, August 2009. Available: `http://bench.cr.yp.to/results-hash.html` (2009/08/26).

[2] I. Damgård. A Design Principle for Hash Functions. In G. Brassard, editor, *Advances in Cryptology – CRYPTO '89, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1990.

[3] D. Gligoroski, V. Klíma, S. J. Knapskog, M. El-Hadedy, J. Amundsen, and S. F. Mjølsnes. Cryptographic Hash Function Blue Midnight Wish. SHA-3 Algorithm Submission, October 2008. Available: `http://people.item.ntnu.no/~danilog/Hash/BMW/Supporting_Documentation/BlueMidnightWishDocumentation.pdf` (2009/04/07).

[4] R. C. Merkle. One Way Hash Functions and DES. In G. Brassard, editor, *Advances in Cryptology – CRYPTO '89, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer, 1990.

[5] National Institute of Standards and Technology. The SHA-3 competition website. Available: `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html` (2009/08/26).

[6] S. S. Thomsen. A near-collision attack on the Blue Midnight Wish compression function. Manuscript, November 2008. Available: `http://www.mat.dtu.dk/people/S.Thomsen/bmw/nc-compress.pdf` (2009/09/09).

A Details of P

P is a permutation that corresponds to first multiplying the input by a matrix $\mathbf{C}$, and then applying reasonably simple permutations to each word. The matrix $\mathbf{C}$ is defined in the same way for all variants of BMW. We have

$$\mathbf{C} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 1 & -1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 1 \\
1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & -1 & 0 \\
0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 1 \\
-1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\
0 & 0 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\
1 & 0 & 0 & -1 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & -1 & 0 & 0 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
-1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 1 & 1 & 0 & 0
\end{bmatrix}.$$

Note that $\mathbf{C}$ is invertible modulo 2^{32} and modulo 2^{64}, but we omit a specification of the inverses.

As an example, if the input is the vector X, and $T = \mathbf{C} \cdot X$, then (e.g.) $T_0 = X_5 - X_7 + X_{10} + X_{13} + X_{14} \bmod 2^w$. The output $Y = P(X)$ is computed as $Y_i \leftarrow s_{i \bmod 5}(T_i)$ for $0 \le i < 16$, where the permutations s_i, $0 \le i \le 4$, are defined in Appendix B.

B Sub-functions used in P and f_1

The sub-functions s_i, $0 \leq i \leq 4$, and r_i, $1 \leq i \leq 7$, used in P and in f_1 are defined as follows.

BMW-224 and BMW-256	BMW-384 and BMW-512
$s_0(x) = x^{\gg 1} \oplus x^{\ll 3} \oplus x^{\lll 4} \oplus x^{\lll 19}$	$s_0(x) = x^{\gg 1} \oplus x^{\ll 3} \oplus x^{\lll 4} \oplus x^{\lll 37}$
$s_1(x) = x^{\gg 1} \oplus x^{\ll 2} \oplus x^{\lll 8} \oplus x^{\lll 23}$	$s_1(x) = x^{\gg 1} \oplus x^{\ll 2} \oplus x^{\lll 13} \oplus x^{\lll 43}$
$s_2(x) = x^{\gg 2} \oplus x^{\ll 1} \oplus x^{\lll 12} \oplus x^{\lll 25}$	$s_2(x) = x^{\gg 2} \oplus x^{\ll 1} \oplus x^{\lll 19} \oplus x^{\lll 53}$
$s_3(x) = x^{\gg 2} \oplus x^{\ll 2} \oplus x^{\lll 15} \oplus x^{\lll 29}$	$s_3(x) = x^{\gg 2} \oplus x^{\ll 2} \oplus x^{\lll 28} \oplus x^{\lll 59}$
$s_4(x) = x^{\gg 1} \oplus x$	$s_4(x) = x^{\gg 1} \oplus x$
$s_5(x) = x^{\gg 2} \oplus x$	$s_5(x) = x^{\gg 2} \oplus x$
$r_1(x) = x^{\lll 3}$	$r_1(x) = x^{\lll 5}$
$r_2(x) = x^{\lll 7}$	$r_2(x) = x^{\lll 11}$
$r_3(x) = x^{\lll 13}$	$r_3(x) = x^{\lll 27}$
$r_4(x) = x^{\lll 16}$	$r_4(x) = x^{\lll 32}$
$r_5(x) = x^{\lll 19}$	$r_5(x) = x^{\lll 37}$
$r_6(x) = x^{\lll 23}$	$r_6(x) = x^{\lll 43}$
$r_7(x) = x^{\lll 27}$	$r_7(x) = x^{\lll 53}$

C Inverses of the matrix B used in f_1

The matrix **B** introduced in Section 2.1.1 is defined as

$$\mathbf{B} = \begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
-1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}.$$

It is circulant, meaning that each row is equal to the row above rotated one position to the right. The inverses modulo 2^{32} and 2^{64} are also circulant. The first row of

$\mathbf{B}^{-1} \bmod 2^{32}$ is (in hexadecimal)

$$[\texttt{abababac}, \texttt{c6c6c6c7}, \texttt{bdbdbdbe}, \texttt{c0c0c0c1}, \texttt{15151515}, \texttt{4e4e4e4e}, \texttt{90909090}, \texttt{cfcfcfd0},$$
$$\texttt{bababab}, \texttt{6c6c6c6d}, \texttt{dbdbdbdc}, \texttt{0c0c0c0c}, \texttt{51515151}, \texttt{e4e4e4e5}, \texttt{09090909}, \texttt{fcfcfcfd}].$$

The first row of $\mathbf{B}^{-1} \bmod 2^{64}$ is

$$[\texttt{abababababababac}, \texttt{c6c6c6c6c6c6c6c7}, \texttt{bdbdbdbdbdbdbdbe}, \texttt{c0c0c0c0c0c0c0c1},$$
$$\texttt{1515151515151515}, \texttt{4e4e4e4e4e4e4e4e}, \texttt{9090909090909090}, \texttt{cfcfcfcfcfcfcfd0},$$
$$\texttt{bababababababab}, \texttt{6c6c6c6c6c6c6c6d}, \texttt{dbdbdbdbdbdbdbdc}, \texttt{0c0c0c0c0c0c0c0c},$$
$$\texttt{5151515151515151}, \texttt{e4e4e4e4e4e4e4e5}, \texttt{0909090909090909}, \texttt{fcfcfcfcfcfcfcfd}].$$