

# Cryptographic Hash Function

## BLUE MIDNIGHT WISH

Norwegian University of Science and Technology

Trondheim, Norway

Danilo Gligoroski

Vlastimil Klima

Svein Johan Knapskog

Mohamed El-Hadedy

Jørn Amundsen

Stig Frode Mjølsnes

October 2008



# Abstract

This is the supporting documentation that describes in details the cryptographic hash function BLUE MIDNIGHT WISH that is submitted as a candidate for SHA-3 hash competition organized by National Institute of Standards and Technology (NIST), according to the public call [1].

# Contents

<b>Cover Page</b>	<b>1</b>
<b>1 Algorithm Specifics</b>	<b>3</b>
1.1 Bit Strings and Integers . . . . .	3
1.2 Parameters, variables and constants . . . . .	4
1.3 General design properties of BLUE MIDNIGHT WISH . . . . .	6
1.4 BLUE MIDNIGHT WISH logic functions . . . . .	7
1.5 Preprocessing . . . . .	7
1.5.1 Padding the message . . . . .	8
BWM224 and BMW256 . . . . .	8
BWM384 and BMW512 . . . . .	9
1.5.2 Parsing the message . . . . .	9
BWM224 and BMW256 . . . . .	9
BWM384 and BMW512 . . . . .	9
1.5.3 Setting the initial double pipe value $H^{(0)}$ . . . . .	10
BWM224 . . . . .	10
BWM256 . . . . .	10
BWM384 . . . . .	10
BWM512 . . . . .	12
<b>2 Description of the Hash Algorithm Blue Midnight Wish</b>	<b>13</b>

## CONTENTS

2.1	Generic description for all variants of the BLUE MIDNIGHT WISH . . . . .	13
2.1.1	BMW224 and BMW256 . . . . .	15
	BMW224 and BMW256 preprocessing . . . . .	17
2.1.2	BMW384 and BMW512 . . . . .	18
	BMW384 and BMW512 preprocessing . . . . .	18
<b>3</b>	<b>Design Rationales</b>	<b>19</b>
3.1	Reasons for default little-endian design . . . . .	19
3.2	Reasons for using double pipe iterative structure . . . . .	19
3.3	Rationales for used constants in BLUE MIDNIGHT WISH . . . . .	20
3.3.1	Constants in logical functions . . . . .	20
3.3.2	Constants in the expansion part . . . . .	20
3.4	Rationales for the bijective “Step 1” in the function $f_0$ . . . . .	21
3.5	Rationales for the bijective “Step 2” in the function $f_0$ . . . . .	23
3.6	Tunable parameters $ExpandRounds_1$ and $ExpandRounds_2$ . . . . .	24
3.6.1	Statements, according to the NIST requirements 2.B.1. . . . .	24
3.7	Cryptanalysis of BLUE MIDNIGHT WISH . . . . .	25
3.7.1	Bijective parts in the compression function of BLUE MIDNIGHT WISH . . . . .	25
3.7.2	Representation as a generalized PGV6 scheme . . . . .	31
3.7.3	Representation as a generalized PGV scheme . . . . .	32
3.7.4	Monomial tests on the block ciphers used in BLUE MIDNIGHT WISH . . . . .	34
3.7.5	Infeasibility of finding collisions, preimages and second preimages . . . . .	38
3.7.6	Approximation of additions and subtractions with XORs . . . . .	39
3.7.7	Cryptanalysis of a scaled down BLUE MIDNIGHT WISH . . . . .	40
3.8	Statements about security, support for applications, HMACs and randomized hashing	45
3.8.1	Security statement according to the NIST requirement 4.A. . . . .	45
3.8.2	Statements according to the NIST requirement 4.A.iii. . . . .	45
3.8.3	Statement about the support of applications . . . . .	45

## CONTENTS

3.8.4	Statement about the special requirements . . . . .	46
3.8.5	Support of HMAC . . . . .	46
3.8.6	BLUE MIDNIGHT WISH support of randomized hashing . . . . .	50
3.8.7	Resistance to SHA-2 attacks . . . . .	50
<b>4</b>	<b>Estimated Computational Efficiency and Memory Requirements</b>	<b>53</b>
4.1	Speed of BLUE MIDNIGHT WISH on NIST SHA-3 Reference Platform . . . . .	53
4.1.1	Speed of the Optimized 32-bit version of BLUE MIDNIGHT WISH . . . . .	53
4.1.2	Speed of the Optimized 64-bit version of BLUE MIDNIGHT WISH . . . . .	54
4.2	Memory requirements of BLUE MIDNIGHT WISH on NIST SHA-3 Reference Platform	54
4.3	Estimates for efficiency and memory requirements on 8-bit processors . . . . .	55
4.4	Estimates for a Compact Hardware Implementation . . . . .	55
4.5	Internal Parallelizability of BLUE MIDNIGHT WISH . . . . .	56
<b>5</b>	<b>Statements</b>	<b>59</b>
5.1	Statement by the Submitter . . . . .	59
5.2	Statement by Patent (and Patent Application) Owner(s) . . . . .	61
5.3	Statement by Reference/Optimized Implementations' Owner(s) . . . . .	62
	<b>References</b>	<b>63</b>

# Cover page

- Name of the submitted algorithm: **BLUE MIDNIGHT WISH**
- Principal submitter's name, e-mail address, telephone, fax, organization, and postal address:
  - Name: Svein Johan Knapskog
  - Email: knapskog@q2s.ntnu.no
  - Tel: +47 735 94328
  - Fax: +47 73 59 27 90
  - Organization: "Centre for Quantifiable Quality of Service in Communication Systems. Centre of Excellence"
  - Address: O.S. Bragstads plass 2E, N-7491 Trondheim, Norway

- Name of the algorithm inventor(s)/developer(s):

## **Inventors:**

- Danilo Gligoroski
- Vlastimil Klima

## **Developers and contributors:**

- Danilo Gligoroski, Norwegian University of Science and Technology, Norway
- Vlastimil Klima, Independent cryptologist - consultant, Czech Republic
- Svein Johan Knapskog, Norwegian University of Science and Technology, Norway
- Mohamed El-Hadedy, Norwegian University of Science and Technology, Norway
- Jøren Amundsen, Norwegian University of Science and Technology, Norway
- Stig Frode Mjølshes, Norwegian University of Science and Technology, Norway

- Name of the owner, if any, of the algorithm:

- Danilo Gligoroski

- Vlastimil Klima

- Signature of the submitter

---

- (optional) Backup point of contact (with telephone, fax, postal address, e-mail address):

- Name: Danilo Gligoroski

- Email: gligoroski@yahoo.com

- Tel: +47 73 59 46 16

- Fax: +47 73 59 69 73

- Organization: Department of Telematics, Faculty of Information Technology, Mathematics and Electrical Engineering, The Norwegian University of Science and Technology (NTNU), O.S. Bragstads plass 2B, N-7491 Trondheim, Norway

# Algorithm Specifics

## 1.1 Bit Strings and Integers

The following terminology related to bit strings, byte strings and integers will be used:

1. A hex digit is an element of the set  $\{0, 1, \dots, 9, A, \dots, F\}$ . A hex digit is the representation of a 4-bit string. For example, the hex digit "7" represents the 4-bit string "0111", and the hex digit "A" represents the 4-bit string "1010".
2. The "little-endian" convention is used when expressing string of bytes stored in memory. That means that beginning from some address "H" if the content of the memory is represented as a 1-byte address increment, then 32-bit and 64-bit integers are expressed as in the example given in Table 1.1. The prefix "0x" is used to annotate that the integer is expressed in hex digit notation.
3. The "big-endian" convention is used when expressing the internal "Bit endianness" for both 32-bit and 64-bit words as integers. That means that within each word, the most significant bit is stored in the left-most bit position. More concretely, a word is a  $w$ -bit string that may be represented as a sequence of hex digits. To convert a word to hex digits, each 4-bit string is converted to its hex digit equivalent. For example, the 32-bit string "1010 0001 0000 0011 1111 1110 0010 0011" has a hexadecimal representation "0xA103FE23" and its value as unsigned long integer is 2701393443. The 64-bit string "1010 0001 0000 0011 1111 1110 0010 0011 0011 0010 1110 1111 0011 0000 0001 1010" has a hexadecimal representation "0xA103FE2332EF301A" and its value as unsigned long long integer is 11602396492168376346.
4. For BLUE MIDNIGHT WISH hash algorithm, the size of  $m$  bits of the message block, depends

Address in memory	Byte value
H	23
H+1	FE
H+2	03
H+3	A1

32-bit integer value: "0xA103FE23"

Address in memory	Byte value
H	1A
H+1	30
H+2	EF
H+3	32
H+4	23
H+5	FE
H+6	03
H+7	A1

64-bit integer value: "0xA103FE2332EF301A"

**Table 1.1:** Default design of the BLUE MIDNIGHT WISH is "Little-endian"

on the variant of algorithm (BMW224, BMW256, BMW384 or BMW512).

- (a) For BMW224 and BMW256, each message block has 512 bits, which are represented as a sequence of sixteen 32-bit words.
- (b) For BMW384 and BMW512, each message block has 1024 bits, which are represented as a sequence of sixteen 64-bit words.

## 1.2 Parameters, variables and constants

The following parameters and variables are used in the specification of BLUE MIDNIGHT WISH :

$H$	Double pipe. It is a the chaining value that is two times wider than the final message digest of $n$ bits.
$Q$	Quadruple pipe.
$H^{(i)}$	The $i$ -th double pipe value. $H^{(0)}$ is the initial double pipe value. $H^{(N)}$ is the final double pipe value and is used to determine the message digest of $n$ bits.
$Q^{(i)}$	The $i$ -th quadruple pipe value.
$H_j^{(i)}$	The $j$ -th word of the $i$ -th double pipe value $H^{(i)}$ , where $H_0^{(i)}$ is the is the left-most word.

$Q_j^{(i)}$	The $j$ -th word of the $i$ -th quadruple pipe value $Q^{(i)}$ , where $Q_0^{(i)}$ is the left-most word.
$Q_a^{(i)}$	The first 16 words from $Q^{(i)}$ , i.e. $Q_a^{(i)} = (Q_0^{(i)}, \dots, Q_{15}^{(i)})$ .
$Q_b^{(i)}$	The last 16 words from $Q^{(i)}$ , i.e. $Q_b^{(i)} = (Q_{16}^{(i)}, \dots, Q_{31}^{(i)})$ .
$k$	Number of zeroes appended to a message during the padding step.
$l$	Length of the message $M$ , in bits.
$m$	Number of bits in a message block, $M^{(i)}$ .
$M$	Message to be hashed.
$M^{(i)}$	Message block $i$ , with a size of $m$ bits.
$M_j^{(i)}$	The $j$ -th word of the $i$ -th message block $M^{(i)}$ , where $M_0^{(i)}$ is the left-most word.
$r$	Number of bits to be rotated or shifted when a word is operated upon.
$N$	Number of blocks in the padded message.
$XL, XH$	Two temporary words (32-bit or 64-bit – depending on the variant of algorithm) used in the computation of the double pipe.
$0x05555555$	A hex digit presentation of a 32-bit constant (unsigned long integer).
$K_j = j \times (0x05555555)$ $j = 16, 17, \dots, 31$	A 32-bit constant (unsigned long) obtained by multiplying the constant $0x05555555$ by an integer $j$ , where $j$ is in the range from 16 to 31.
$0x0555555555555555$	A hex digit presentation of a 64-bit constant (unsigned long long integer).

$K_j = j \times (0x0555555555555555)$   
 $j = 16, 17, \dots, 31$

A 64-bit constant (unsigned long long) obtained by multiplying the constant  $0x0555555555555555$  by an integer  $j$ , where  $j$  is in the range from 16 to 31.

$ExpandRounds_1 = 2,$   
 $ExpandRounds_2 = 14$

Two tunable parameters that determine what expansion function and how many times will be used in the part of double pipe expansion. These two parameters are connected by the relation

$$ExpandRounds_1 + ExpandRounds_2 = 16$$

### 1.3 General design properties of BLUE MIDNIGHT WISH

BLUE MIDNIGHT WISH follows the general design pattern that is common for most of the known hash functions. It means that it has two stages (and several sub-stages within every stage):

1. Preprocessing
  - (a) padding a message,
  - (b) parsing the padded message into  $m$ -bit blocks, and
  - (c) setting initialization values to be used in the hash computation.
2. Hash computation
  - (a) generating a message schedule from the padded message,
  - (b) using that schedule, along with functions, constants, and word operations to iteratively generate a series of double pipe values,
  - (c) The  $n$  Least Significant Bits (LSB) of the final double pipe value generated by the hash computation are used to determine the message digest.

Depending on the context we will sometimes refer to the hash function as BLUE MIDNIGHT WISH and sometimes as BMW224, BMW256, BMW384 or BMW512.

In Table 1.2, we give the basic properties of all four variants of the BLUE MIDNIGHT WISH hash algorithms.

The following operations are applied in BLUE MIDNIGHT WISH :

Algorithm abbreviation	Message size $l$ (in bits)	Block size $m$ (in bits)	Word size $w$ (in bits)	Endianness	Digest size $n$ (in bits)	Support of "one-pass" streaming mode
BMW224	$< 2^{64}$	512	32	Little-endian	224	Yes
BMW256	$< 2^{64}$	512	32	Little-endian	256	Yes
BMW384	$< 2^{64}$	1024	64	Little-endian	384	Yes
BMW512	$< 2^{64}$	1024	64	Little-endian	512	Yes

**Table 1.2:** Basic properties of all four variants of the BLUE MIDNIGHT WISH

1. Bitwise logic word operations:  $\wedge$  – AND,  $\vee$  – OR and  $\oplus$  – XOR.
2. Addition  $+$  and subtraction  $-$  modulo  $2^{32}$  or modulo  $2^{64}$ .
3. Shift right operation,  $SHR^r(x)$ , where  $x$  is a 32-bit or 64-bit word and  $r$  is an integer with  $0 < r < 32$  (resp.  $0 < r < 64$ ).
4. Shift left operation,  $SHL^r(x)$ , where  $x$  is a 32-bit or 64-bit word and  $r$  is an integer with  $0 < r < 32$  (resp.  $0 < r < 64$ ).
5. Rotate left (circular left shift) operation,  $ROTL^r(x)$ , where  $x$  is a 32-bit or 64-bit word and  $r$  is an integer with  $0 < r < 32$  (resp.  $0 < r < 64$ ).

## 1.4 BLUE MIDNIGHT WISH logic functions

BLUE MIDNIGHT WISH uses the logic functions, summarized in Table 1.3.

## 1.5 Preprocessing

Preprocessing consists of three steps:

1. padding the message  $M$ ,
2. parsing the padded message into message blocks, and
3. setting the initial double pipe value,  $H^{(0)}$ .

BMW224/BMW256	BMW384/BMW512
$s_0(x) = SHR^1(x) \oplus SHL^1(x) \oplus ROTL^4(x) \oplus ROTL^{19}(x)$ $s_1(x) = SHR^1(x) \oplus SHL^2(x) \oplus ROTL^8(x) \oplus ROTL^{23}(x)$ $s_2(x) = SHR^2(x) \oplus SHL^1(x) \oplus ROTL^{12}(x) \oplus ROTL^{25}(x)$ $s_3(x) = SHR^2(x) \oplus SHL^2(x) \oplus ROTL^{15}(x) \oplus ROTL^{29}(x)$ $s_4(x) = SHR^1(x) \oplus x$ $s_5(x) = SHR^2(x) \oplus x$ $r_1(x) = ROTL^3(x)$ $r_2(x) = ROTL^7(x)$ $r_3(x) = ROTL^{13}(x)$ $r_4(x) = ROTL^{16}(x)$ $r_5(x) = ROTL^{19}(x)$ $r_6(x) = ROTL^{23}(x)$ $r_7(x) = ROTL^{27}(x)$	$s_0(x) = SHR^1(x) \oplus SHL^1(x) \oplus ROTL^4(x) \oplus ROTL^{37}(x)$ $s_1(x) = SHR^1(x) \oplus SHL^2(x) \oplus ROTL^{13}(x) \oplus ROTL^{43}(x)$ $s_2(x) = SHR^2(x) \oplus SHL^1(x) \oplus ROTL^{19}(x) \oplus ROTL^{53}(x)$ $s_3(x) = SHR^2(x) \oplus SHL^2(x) \oplus ROTL^{28}(x) \oplus ROTL^{59}(x)$ $s_4(x) = SHR^1(x) \oplus x$ $s_5(x) = SHR^2(x) \oplus x$ $r_1(x) = ROTL^5(x)$ $r_2(x) = ROTL^{11}(x)$ $r_3(x) = ROTL^{27}(x)$ $r_4(x) = ROTL^{32}(x)$ $r_5(x) = ROTL^{37}(x)$ $r_6(x) = ROTL^{43}(x)$ $r_7(x) = ROTL^{53}(x)$
$expand_1(j) = s_1(Q_{j-16}^{(i)} + s_2(Q_{j-15}^{(i)} + s_3(Q_{j-14}^{(i)} + s_0(Q_{j-13}^{(i)} + s_1(Q_{j-12}^{(i)} + s_2(Q_{j-11}^{(i)} + s_3(Q_{j-10}^{(i)} + s_0(Q_{j-9}^{(i)} + s_1(Q_{j-8}^{(i)} + s_2(Q_{j-7}^{(i)} + s_3(Q_{j-6}^{(i)} + s_0(Q_{j-5}^{(i)} + s_1(Q_{j-4}^{(i)} + s_2(Q_{j-3}^{(i)} + s_3(Q_{j-2}^{(i)} + s_0(Q_{j-1}^{(i)} + M_{(j-16) \bmod 16}^{(i)} + M_{(j-13) \bmod 16}^{(i)} - M_{(j-6) \bmod 16}^{(i)} + K_j$	$expand_1(j) = s_1(Q_{j-16}^{(i)} + s_2(Q_{j-15}^{(i)} + s_3(Q_{j-14}^{(i)} + s_0(Q_{j-13}^{(i)} + s_1(Q_{j-12}^{(i)} + s_2(Q_{j-11}^{(i)} + s_3(Q_{j-10}^{(i)} + s_0(Q_{j-9}^{(i)} + s_1(Q_{j-8}^{(i)} + s_2(Q_{j-7}^{(i)} + s_3(Q_{j-6}^{(i)} + s_0(Q_{j-5}^{(i)} + s_1(Q_{j-4}^{(i)} + s_2(Q_{j-3}^{(i)} + s_3(Q_{j-2}^{(i)} + s_0(Q_{j-1}^{(i)} + M_{(j-16) \bmod 16}^{(i)} + M_{(j-13) \bmod 16}^{(i)} - M_{(j-6) \bmod 16}^{(i)} + K_j$
$expand_2(j) = Q_{j-16}^{(i)} + r_1(Q_{j-15}^{(i)} + Q_{j-14}^{(i)} + r_2(Q_{j-13}^{(i)} + Q_{j-12}^{(i)} + r_3(Q_{j-11}^{(i)} + Q_{j-10}^{(i)} + r_4(Q_{j-9}^{(i)} + Q_{j-8}^{(i)} + r_5(Q_{j-7}^{(i)} + Q_{j-6}^{(i)} + r_6(Q_{j-5}^{(i)} + Q_{j-4}^{(i)} + r_7(Q_{j-3}^{(i)} + s_5(Q_{j-2}^{(i)} + s_4(Q_{j-1}^{(i)} + M_{(j-16) \bmod 16}^{(i)} + M_{(j-13) \bmod 16}^{(i)} - M_{(j-6) \bmod 16}^{(i)} + K_j$	$expand_2(j) = Q_{j-16}^{(i)} + r_1(Q_{j-15}^{(i)} + Q_{j-14}^{(i)} + r_2(Q_{j-13}^{(i)} + Q_{j-12}^{(i)} + r_3(Q_{j-11}^{(i)} + Q_{j-10}^{(i)} + r_4(Q_{j-9}^{(i)} + Q_{j-8}^{(i)} + r_5(Q_{j-7}^{(i)} + Q_{j-6}^{(i)} + r_6(Q_{j-5}^{(i)} + Q_{j-4}^{(i)} + r_7(Q_{j-3}^{(i)} + s_5(Q_{j-2}^{(i)} + s_4(Q_{j-1}^{(i)} + M_{(j-16) \bmod 16}^{(i)} + M_{(j-13) \bmod 16}^{(i)} - M_{(j-6) \bmod 16}^{(i)} + K_j$

Table 1.3: Logic functions used in BLUE MIDNIGHT WISH

### 1.5.1 Padding the message

The message  $M$ , shall be padded before hash computation begins. The purpose of this padding is to ensure that the padded message is a multiple of 512 or 1024 bits, depending on the size of the message digest  $n$ .

#### BWM224 and BMW256

Suppose that the length of the message  $M$  is  $l$  bits. Append the bit "1" to the end of the message, followed by  $k$  zero bits, where  $k$  is the smallest, non-negative solution to the equation  $l + 1 + k \equiv 448 \pmod{512}$ . Then append the 64-bit block that is equal to the number  $l$  expressed using a binary representation. For example, the (8-bit ASCII) message "abc" has length  $8 \times 3 = 24$ , so the message is padded with the bit "1", then  $448 - (24 + 1) = 423$  zero bits, and then the 64-bit binary

presentation of the number 24, to become the 512-bit padded message.

$$\underbrace{01100001}_{\text{"a"}} \underbrace{01100010}_{\text{"b"}} \underbrace{01100011}_{\text{"c"}} 1 \overbrace{00 \dots 00}^{423} \overbrace{00 \dots 011000}^{64} \\ l=24$$

### BWM384 and BMW512

Suppose that the length of the message  $M$  is  $l$  bits. Append the bit "1" to the end of the message, followed by  $k$  zero bits, where  $k$  is the smallest, non-negative solution to the equation  $l + 1 + k \equiv 960 \pmod{1024}$ . Then append the 64-bit block that is equal to the number  $l$  expressed using a binary representation. For example, the (8-bit ASCII) message "abc" has length  $8 \times 3 = 24$ , so the message is padded with the bit "1", then  $960 - (24 + 1) = 935$  zero bits, and then the 64-bit binary presentation of the number 24, to become the 1024-bit padded message.

$$\underbrace{01100001}_{\text{"a"}} \underbrace{01100010}_{\text{"b"}} \underbrace{01100011}_{\text{"c"}} 1 \overbrace{00 \dots 00}^{935} \overbrace{00 \dots 011000}^{64} \\ l=24$$

### 1.5.2 Parsing the message

After a message has been padded, it must be parsed into  $N$   $m$ -bit blocks before the hash computation can begin.

### BWM224 and BMW256

For BMW224 and BMW256, the padded message is parsed into  $N$  512-bit blocks,  $M^{(1)}, M^{(2)}, \dots, M^{(N)}$ . Since the 512 bits of the input block may be expressed as sixteen 32-bit words, the first 32 bits of message block  $i$  are denoted  $M_0^{(i)}$ , the next 32 bits are  $M_1^{(i)}$ , and so on up to  $M_{15}^{(i)}$ .

### BWM384 and BMW512

For BMW384 and BMW512, the padded message is parsed into  $N$  1024-bit blocks,  $M^{(1)}, M^{(2)}, \dots, M^{(N)}$ . Since the 1024 bits of the input block may be expressed as sixteen 64-bit words, the first 64 bits of message block  $i$  are denoted  $M_0^{(i)}$ , the next 64 bits are  $M_1^{(i)}$ , and so on up to  $M_{15}^{(i)}$ .

### 1.5.3 Setting the initial double pipe value $H^{(0)}$

Before hash computation begins for each of the hash algorithms, the initial double pipe value  $H^{(0)}$  must be set. The size and the value of words in  $H^{(0)}$  depends on the message digest size  $n$ . As it is shown in the following subsections the constants consist of concatenation of the consecutive 8-bit natural numbers. Since BMW224 is the same as BMW256 except for the final truncation, they have to have different initial values. Thus, the initial double pipe of BMW224 starts from the byte value 0x00 and takes all 64 successive byte values up to the value 0x3F. Then, the initial double pipe of BMW256 starts from the byte value 0x40 and takes all 64 successive byte values up to the value 0x7F. The same situation is with BMW384 and BMW512, but since there the variables are 64-bit long, the initial double pipe of BMW384 starts from the byte value 0x00 and takes all 128 successive byte values up to the value 0x7F and the initial double pipe of BMW512 starts from the byte value 0x80 and takes all 128 successive byte values up to the value 0xFF.

#### BWM224

For BMW224, the initial double pipe value  $H^{(0)}$  shall consist of sixteen 32-bit words given in Table 1.4.

$H_0^{(0)} = 0x00010203$	$H_1^{(0)} = 0x04050607$
$H_2^{(0)} = 0x08090A0B$	$H_3^{(0)} = 0x0C0D0E0F$
$H_4^{(0)} = 0x10111213$	$H_5^{(0)} = 0x14151617$
$H_6^{(0)} = 0x18191A1B$	$H_7^{(0)} = 0x1C1D1E1F$
$H_8^{(0)} = 0x20212223$	$H_9^{(0)} = 0x24252627$
$H_{10}^{(0)} = 0x28292A2B$	$H_{11}^{(0)} = 0x2C2D2E2F$
$H_{12}^{(0)} = 0x30313233$	$H_{13}^{(0)} = 0x34353637$
$H_{14}^{(0)} = 0x38393A3B$	$H_{15}^{(0)} = 0x3C3D3E3F$

**Table 1.4:** Initial double pipe  $H^{(0)}$  for BMW224

#### BWM256

For BMW256, the initial double pipe value  $H^{(0)}$  shall consist of sixteen 32-bit words given in Table 1.5.

$H_0^{(0)} = 0x40414243$	$H_1^{(0)} = 0x44454647$
$H_2^{(0)} = 0x48494A4B$	$H_3^{(0)} = 0x4C4D4E4F$
$H_4^{(0)} = 0x50515253$	$H_5^{(0)} = 0x54555657$
$H_6^{(0)} = 0x58595A5B$	$H_7^{(0)} = 0x5C5D5E5F$
$H_8^{(0)} = 0x60616263$	$H_9^{(0)} = 0x64656667$
$H_{10}^{(0)} = 0x68696A6B$	$H_{11}^{(0)} = 0x6C6D6E6F$
$H_{12}^{(0)} = 0x70717273$	$H_{13}^{(0)} = 0x74757677$
$H_{14}^{(0)} = 0x78797A7B$	$H_{15}^{(0)} = 0x7C7D7E7F$

**Table 1.5:** Initial double pipe  $H^{(0)}$  for BMW256

### BWM384

For BMW384, the initial double pipe value  $H^{(0)}$  shall consist of sixteen 64-bit words given in Table 1.6.

$H_0^{(0)} = 0x0001020304050607$	$H_1^{(0)} = 0x08090A0B0C0D0E0F$
$H_2^{(0)} = 0x1011121314151617$	$H_3^{(0)} = 0x18191A1B1C1D1E1F$
$H_4^{(0)} = 0x2021222324252627$	$H_5^{(0)} = 0x28292A2B2C2D2E2F$
$H_6^{(0)} = 0x3031323324353637$	$H_7^{(0)} = 0x38393A3B3C3D3E3F$
$H_8^{(0)} = 0x4041424344454647$	$H_9^{(0)} = 0x48494A4B4C4D4E4F$
$H_{10}^{(0)} = 0x5051525354555657$	$H_{11}^{(0)} = 0x58595A5B5C5D5E5F$
$H_{12}^{(0)} = 0x6061626364656667$	$H_{13}^{(0)} = 0x68696A6B6C6D6E6F$
$H_{14}^{(0)} = 0x7071727374757677$	$H_{15}^{(0)} = 0x78797A7B7C7D7E7F$

**Table 1.6:** Initial double pipe  $H^{(0)}$  for BMW384

**BWM512**

For BMW512, the initial double pipe value  $H^{(0)}$  shall consist of sixteen 64-bit words given in Table 1.7.

$H_0^{(0)} = 0x8081828384858687$	$H_1^{(0)} = 0x88898A8B8C8D8E8F$
$H_2^{(0)} = 0x9091929394959697$	$H_3^{(0)} = 0x98999A9B9C9D9E9F$
$H_4^{(0)} = 0xA0A1A2A3A4A5A6A7$	$H_5^{(0)} = 0xA8A9AAABACADAEAF$
$H_6^{(0)} = 0xB0B1B2B3B4B5B6B7$	$H_7^{(0)} = 0xB8B9BABBBBCBDBEBF$
$H_8^{(0)} = 0xC0C1C2C3C4C5C6C7$	$H_9^{(0)} = 0xC8C9CACBCCCDCECF$
$H_{10}^{(0)} = 0xD0D1D2D3D4D5D6D7$	$H_{11}^{(0)} = 0xD8D9DADBDCDDDEDF$
$H_{12}^{(0)} = 0xE0E1E2E3E4E5E6E7$	$H_{13}^{(0)} = 0xE8E9EAEBECEDEEEF$
$H_{14}^{(0)} = 0xF0F1F2F3F4F5F6F7$	$H_{15}^{(0)} = 0xF8F9FAFBFCFDFF$

**Table 1.7:** Initial double pipe  $H^{(0)}$  for BMW384

# Description of the Hash Algorithm Blue Midnight Wish

## 2.1 Generic description for all variants of the BLUE MIDNIGHT WISH

First we are giving a generic description for all variants of the BLUE MIDNIGHT WISH hash algorithm. Then, in the following subsections we will give a detailed functional description for the specific variants of the BLUE MIDNIGHT WISH hash algorithm (for four different message digest sizes:  $n = 224$ ,  $n = 256$ ,  $n = 384$  and  $n = 512$  bits).

In the generic description we are using three functions:

1. The first function is  $f_0 : \{0, 1\}^{2m} \rightarrow \{0, 1\}^m$ . It takes two arguments  $M^{(i)}$  and  $H^{(i-1)}$  each of  $m$  bits and bijectively transforms  $M^{(i)} \oplus H^{(i-1)}$ . Here,  $M^{(i)}$  is the  $i$ -th message block and  $H^{(i-1)}$  is the current value of the double pipe. The result  $Q_a^{(i)} = f_0(M^{(i)}, H^{(i-1)}) = \mathbf{A}_2(\mathbf{A}_1(M^{(i)} \oplus H^{(i-1)}))$ , is the first part of the extended (quadruple) pipe. The concrete definition of bijections  $\mathbf{A}_1, \mathbf{A}_2 : \{0, 1\}^m \rightarrow \{0, 1\}^m$  will be given later. We denote  $Q_a^{(i)} = (Q_0^{(i)}, \dots, Q_{15}^{(i)})$ . **Note:** There is a small inconsistency in the notation of  $f_0$  as a function of  $2m$  bits  $f_0(M^{(i)}, H^{(i-1)})$  and as a function of  $m$  bits -  $f_0(M^{(i)} \oplus H^{(i-1)})$ . In the following text we treat  $f_0(M^{(i)}, H^{(i-1)})$  as an extended notation of the expression  $f_0(M^{(i)} \oplus H^{(i-1)})$ . We will use both expressions in different contexts.
2. The second function  $f_1$  takes also two arguments: a message block  $M^{(i)}$  of  $m$  bits and the value of  $Q_a^{(i)}$  of  $m$  bits, to produce the second part  $Q_b^{(i)}$  of the extended (quadrupled) pipe  $Q^{(i)}$ . The function can be briefly described as  $f_1 : \{0, 1\}^{2m} \rightarrow \{0, 1\}^m$ , and  $Q_b^{(i)} = f_1(M^{(i)}, Q_a^{(i)})$ .

<b>Algorithm: BLUE MIDNIGHT WISH</b>
<b>Input:</b> Message $M$ of length $l$ bits, and the message digest size $n$ .
<b>Output:</b> A message digest $Hash$ , that is long $n$ bits.
<ol style="list-style-type: none"> <li>1. Preprocessing <ol style="list-style-type: none"> <li>(a) Pad the message <math>M</math>.</li> <li>(b) Parse the padded message into <math>N</math>, <math>m</math>-bit message blocks, <math>M^{(1)}, M^{(2)}, \dots, M^{(N)}</math>.</li> <li>(c) Set the initial value of the double pipe <math>H^{(0)}</math>.</li> </ol> </li> <li>2. Hash computation <p style="margin-left: 2em;">For <math>i = 1</math> to <math>N</math></p> <p style="margin-left: 2em;">{</p> <p style="margin-left: 4em;"><math>Q_a^{(i)} = f_0(M^{(i)}, H^{(i-1)});</math></p> <p style="margin-left: 4em;"><math>Q_b^{(i)} = f_1(M^{(i)}, Q_a^{(i)});</math></p> <p style="margin-left: 4em;"><math>H^{(i)} = f_2(M^{(i)}, Q_a^{(i)}, Q_b^{(i)});</math></p> <p style="margin-left: 2em;">}</p> </li> <li>3. <math>Hash = \text{Take\_}n\_\text{Least\_Significant\_Bits}(H^{(N)})</math>.</li> </ol>

**Table 2.1:** A generic description of the BLUE MIDNIGHT WISH hash algorithm

3. For the third function  $f_2$  we are using the term *folding* to describe its mapping property to map  $3m$  bits to  $m$  bits. It takes two arguments: a message block  $M^{(i)}$  of  $m$  bits and the current value of the extended pipe  $Q^{(i)} = (Q_a^{(i)}, Q_b^{(i)})$  which has  $2m$  bits, to produce a new double pipe  $H^{(i)}$  of  $m$  bits. So,  $f_2 : \{0, 1\}^{3m} \rightarrow \{0, 1\}^m$  and  $H^{(i)} = f_2(M^{(i)}, Q^{(i)}) \equiv f_2(M^{(i)}, Q_a^{(i)}, Q_b^{(i)})$ .

The generic description of BLUE MIDNIGHT WISH hash algorithm is given in Table 2.1. A graphic presentation of the Blue Midnight Wish hash algorithm is given in the Figure 2.1 and its compression function is given in the Figure 2.2.

The function  $f_0 : \{0, 1\}^{2m} \rightarrow \{0, 1\}^m$  is defined in the Table 2.2.

The function  $f_1 : \{0, 1\}^{2m} \rightarrow \{0, 1\}^m$  is defined in the Table 2.3.

The function  $f_2 : \{0, 1\}^{3m} \rightarrow \{0, 1\}^m$  is defined in the Table 2.4.

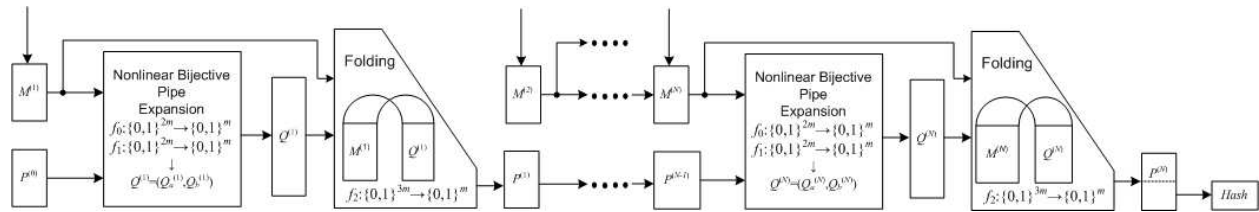


Figure 2.1: A graphic presentation of the BLUE MIDNIGHT WISH hash algorithm.

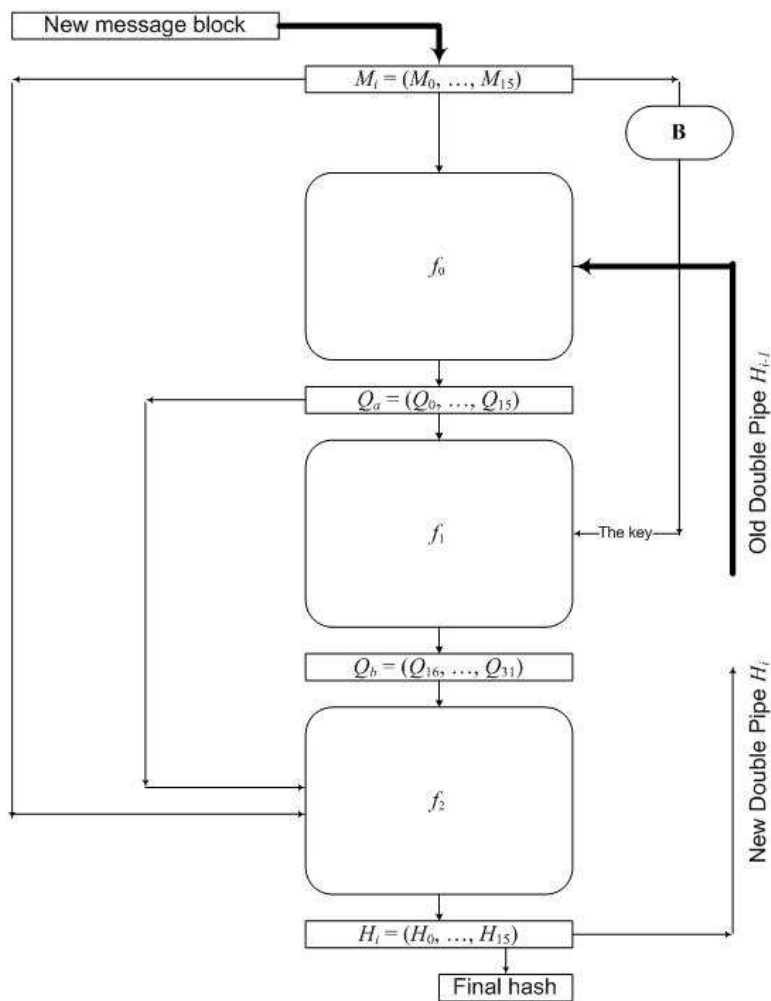


Figure 2.2: Graphical presentation of the compression function in BLUE MIDNIGHT WISH

### 2.1.1 BMW224 and BMW256

BMW224 and BMW256 may be used to hash a message  $M$ , having a length of  $l$  bits, where  $0 \leq l < 2^{64}$ . The algorithms use

1. sixteen 32-bit working variables that are part of the double pipe, and

$f_0 : \{0, 1\}^{2m} \rightarrow \{0, 1\}^m$			
<b>Input:</b> Message block $M^{(i)} = (M_0^{(i)}, M_1^{(i)}, \dots, M_{15}^{(i)})$ , and the previous double pipe $H^{(i-1)} = (H_0^{(i-1)}, H_1^{(i-1)}, \dots, H_{15}^{(i-1)})$ .			
<b>Output:</b> First part of the quadruple pipe $Q_a^{(i)} = (Q_0^{(i)}, Q_1^{(i)}, \dots, Q_{15}^{(i)})$ .			
1. Bijective transform of $M^{(i)} \oplus H^{(i-1)}$ :			
$W_0^{(i)}$	$=$	$(M_5^{(i)} \oplus H_5^{(i-1)}) - (M_7^{(i)} \oplus H_7^{(i-1)}) + (M_{10}^{(i)} \oplus H_{10}^{(i-1)}) + (M_{13}^{(i)} \oplus H_{13}^{(i-1)}) + (M_{14}^{(i)} \oplus H_{14}^{(i-1)})$	
$W_1^{(i)}$	$=$	$(M_6^{(i)} \oplus H_6^{(i-1)}) - (M_8^{(i)} \oplus H_8^{(i-1)}) + (M_{11}^{(i)} \oplus H_{11}^{(i-1)}) + (M_{14}^{(i)} \oplus H_{14}^{(i-1)}) - (M_{15}^{(i)} \oplus H_{15}^{(i-1)})$	
$W_2^{(i)}$	$=$	$(M_0^{(i)} \oplus H_0^{(i-1)}) + (M_7^{(i)} \oplus H_7^{(i-1)}) + (M_9^{(i)} \oplus H_9^{(i-1)}) - (M_{12}^{(i)} \oplus H_{12}^{(i-1)}) + (M_{15}^{(i)} \oplus H_{15}^{(i-1)})$	
$W_3^{(i)}$	$=$	$(M_0^{(i)} \oplus H_0^{(i-1)}) - (M_1^{(i)} \oplus H_1^{(i-1)}) + (M_8^{(i)} \oplus H_8^{(i-1)}) - (M_{10}^{(i)} \oplus H_{10}^{(i-1)}) + (M_{13}^{(i)} \oplus H_{13}^{(i-1)})$	
$W_4^{(i)}$	$=$	$(M_1^{(i)} \oplus H_1^{(i-1)}) + (M_2^{(i)} \oplus H_2^{(i-1)}) + (M_9^{(i)} \oplus H_9^{(i-1)}) - (M_{11}^{(i)} \oplus H_{11}^{(i-1)}) - (M_{14}^{(i)} \oplus H_{14}^{(i-1)})$	
$W_5^{(i)}$	$=$	$(M_3^{(i)} \oplus H_3^{(i-1)}) - (M_2^{(i)} \oplus H_2^{(i-1)}) + (M_{10}^{(i)} \oplus H_{10}^{(i-1)}) - (M_{12}^{(i)} \oplus H_{12}^{(i-1)}) + (M_{13}^{(i)} \oplus H_{13}^{(i-1)})$	
$W_6^{(i)}$	$=$	$(M_4^{(i)} \oplus H_4^{(i-1)}) - (M_0^{(i)} \oplus H_0^{(i-1)}) - (M_3^{(i)} \oplus H_3^{(i-1)}) - (M_{11}^{(i)} \oplus H_{11}^{(i-1)}) + (M_{13}^{(i)} \oplus H_{13}^{(i-1)})$	
$W_7^{(i)}$	$=$	$(M_1^{(i)} \oplus H_1^{(i-1)}) - (M_4^{(i)} \oplus H_4^{(i-1)}) - (M_5^{(i)} \oplus H_5^{(i-1)}) - (M_{12}^{(i)} \oplus H_{12}^{(i-1)}) - (M_{14}^{(i)} \oplus H_{14}^{(i-1)})$	
$W_8^{(i)}$	$=$	$(M_2^{(i)} \oplus H_2^{(i-1)}) - (M_5^{(i)} \oplus H_5^{(i-1)}) - (M_6^{(i)} \oplus H_6^{(i-1)}) + (M_{13}^{(i)} \oplus H_{13}^{(i-1)}) - (M_{15}^{(i)} \oplus H_{15}^{(i-1)})$	
$W_9^{(i)}$	$=$	$(M_0^{(i)} \oplus H_0^{(i-1)}) - (M_3^{(i)} \oplus H_3^{(i-1)}) + (M_6^{(i)} \oplus H_6^{(i-1)}) - (M_7^{(i)} \oplus H_7^{(i-1)}) + (M_{14}^{(i)} \oplus H_{14}^{(i-1)})$	
$W_{10}^{(i)}$	$=$	$(M_8^{(i)} \oplus H_8^{(i-1)}) - (M_1^{(i)} \oplus H_1^{(i-1)}) - (M_4^{(i)} \oplus H_4^{(i-1)}) - (M_7^{(i)} \oplus H_7^{(i-1)}) + (M_{15}^{(i)} \oplus H_{15}^{(i-1)})$	
$W_{11}^{(i)}$	$=$	$(M_8^{(i)} \oplus H_8^{(i-1)}) - (M_0^{(i)} \oplus H_0^{(i-1)}) - (M_2^{(i)} \oplus H_2^{(i-1)}) - (M_5^{(i)} \oplus H_5^{(i-1)}) + (M_9^{(i)} \oplus H_9^{(i-1)})$	
$W_{12}^{(i)}$	$=$	$(M_1^{(i)} \oplus H_1^{(i-1)}) + (M_3^{(i)} \oplus H_3^{(i-1)}) - (M_6^{(i)} \oplus H_6^{(i-1)}) - (M_9^{(i)} \oplus H_9^{(i-1)}) + (M_{10}^{(i)} \oplus H_{10}^{(i-1)})$	
$W_{13}^{(i)}$	$=$	$(M_2^{(i)} \oplus H_2^{(i-1)}) + (M_4^{(i)} \oplus H_4^{(i-1)}) + (M_6^{(i)} \oplus H_6^{(i-1)}) + (M_{10}^{(i)} \oplus H_{10}^{(i-1)}) + (M_{11}^{(i)} \oplus H_{11}^{(i-1)})$	
$W_{14}^{(i)}$	$=$	$(M_3^{(i)} \oplus H_3^{(i-1)}) - (M_5^{(i)} \oplus H_5^{(i-1)}) + (M_8^{(i)} \oplus H_8^{(i-1)}) - (M_{11}^{(i)} \oplus H_{11}^{(i-1)}) - (M_{12}^{(i)} \oplus H_{12}^{(i-1)})$	
$W_{15}^{(i)}$	$=$	$(M_{12}^{(i)} \oplus H_{12}^{(i-1)}) - (M_4^{(i)} \oplus H_4^{(i-1)}) - (M_6^{(i)} \oplus H_6^{(i-1)}) - (M_9^{(i)} \oplus H_9^{(i-1)}) + (M_{13}^{(i)} \oplus H_{13}^{(i-1)})$	
2. Further bijective transform of $W_j^{(i)}, j = 0, \dots, 15$ :			
$Q_0^{(i)}$	$=$	$s_0(W_0^{(i)});$	$Q_3^{(i)} = s_3(W_3^{(i)});$
$Q_1^{(i)}$	$=$	$s_1(W_1^{(i)});$	$Q_6^{(i)} = s_1(W_6^{(i)});$
$Q_2^{(i)}$	$=$	$s_2(W_2^{(i)});$	$Q_7^{(i)} = s_2(W_7^{(i)});$
$Q_4^{(i)}$	$=$	$s_4(W_4^{(i)});$	$Q_8^{(i)} = s_3(W_8^{(i)});$
$Q_5^{(i)}$	$=$	$s_0(W_5^{(i)});$	$Q_9^{(i)} = s_4(W_9^{(i)});$
$Q_6^{(i)}$	$=$	$s_1(W_6^{(i)});$	$Q_{10}^{(i)} = s_0(W_{10}^{(i)});$
$Q_7^{(i)}$	$=$	$s_2(W_7^{(i)});$	$Q_{11}^{(i)} = s_1(W_{11}^{(i)});$
$Q_8^{(i)}$	$=$	$s_3(W_8^{(i)});$	$Q_{12}^{(i)} = s_2(W_{12}^{(i)});$
$Q_9^{(i)}$	$=$	$s_4(W_9^{(i)});$	$Q_{13}^{(i)} = s_3(W_{13}^{(i)});$
$Q_{10}^{(i)}$	$=$	$s_0(W_{10}^{(i)});$	$Q_{14}^{(i)} = s_4(W_{14}^{(i)});$
$Q_{11}^{(i)}$	$=$	$s_1(W_{11}^{(i)});$	$Q_{15}^{(i)} = s_0(W_{15}^{(i)});$
$Q_{12}^{(i)}$	$=$	$s_2(W_{12}^{(i)});$	
$Q_{13}^{(i)}$	$=$	$s_3(W_{13}^{(i)});$	
$Q_{14}^{(i)}$	$=$	$s_4(W_{14}^{(i)});$	
$Q_{15}^{(i)}$	$=$	$s_0(W_{15}^{(i)});$	

**Table 2.2:** Definition of the function  $f_0$  of BLUE MIDNIGHT WISH

$f_1 : \{0, 1\}^{2m} \rightarrow \{0, 1\}^m$			
<b>Input:</b> Message block $M^{(i)} = (M_0^{(i)}, M_1^{(i)}, \dots, M_{15}^{(i)})$ , and the first part of quadruple pipe $Q_a^{(i)} = (Q_0^{(i)}, Q_1^{(i)}, \dots, Q_{15}^{(i)})$ .			
<b>Output:</b> Second part of the quadruple pipe $Q_b^{(i)} = (Q_{16}^{(i)}, Q_{17}^{(i)}, \dots, Q_{31}^{(i)})$ .			
1. Double pipe expansion according to the tunable parameters $ExpandRounds_1$ and $ExpandRounds_2$ .			
1.1 For $ii = 0$ to $ExpandRounds_1 - 1$			
	$Q_{ii+16}^{(i)}$	$=$	$expand_1(ii + 16)$
1.2 For $ii = ExpandRounds_1$ to $ExpandRounds_1 + ExpandRounds_2 - 1$			
	$Q_{ii+16}^{(i)}$	$=$	$expand_2(ii + 16)$

**Table 2.3:** Definition of the function  $f_1$  of BLUE MIDNIGHT WISH

2. additional sixteen 32-bit working variables that together with the variables of the double pipe, make the extended (quadruple) pipe.

The words of the quadruple pipe are labeled  $Q_0^{(i)}, Q_1^{(i)}, \dots, Q_{31}^{(i)}$ . The words of the initial double pipe are labeled  $H_0^{(i-1)}, H_1^{(i-1)}, \dots, H_{15}^{(i-1)}$ , which will hold the initial double pipe value  $H^{(0)}$ , re-

<b>Folding</b> $f_2: \{0,1\}^{3m} \rightarrow \{0,1\}^m$	
<b>Input:</b> Message block $M^{(i)} = (M_0^{(i)}, M_1^{(i)}, \dots, M_{15}^{(i)})$ , quadruple pipe $Q^{(i)} = (Q_0^{(i)}, Q_1^{(i)}, \dots, Q_{15}^{(i)}, Q_{16}^{(i)}, \dots, Q_{31}^{(i)})$ .	
<b>Output:</b> New double pipe $H^{(i)} = (H_0^{(i)}, H_1^{(i)}, \dots, H_{15}^{(i)})$ .	
1. Compute cumulative temporary variables $XL$ and $XH$ . $XL = Q_{16}^{(i)} \oplus Q_{17}^{(i)} \oplus \dots \oplus Q_{23}^{(i)}$ $XH = XL \oplus Q_{24}^{(i)} \oplus Q_{25}^{(i)} \oplus \dots \oplus Q_{31}^{(i)}$	
2. Compute the new double pipe $H^{(i)}$ : $H_0^{(i)} = (SHL^5(XH) \oplus SHR^5(Q_{16}^{(i)}) \oplus M_0^{(i)}) + (XL \oplus Q_{24}^{(i)} \oplus Q_0^{(i)})$ $H_1^{(i)} = (SHR^7(XH) \oplus SHL^8(Q_{17}^{(i)}) \oplus M_1^{(i)}) + (XL \oplus Q_{25}^{(i)} \oplus Q_1^{(i)})$ $H_2^{(i)} = (SHR^5(XH) \oplus SHL^5(Q_{18}^{(i)}) \oplus M_2^{(i)}) + (XL \oplus Q_{26}^{(i)} \oplus Q_2^{(i)})$ $H_3^{(i)} = (SHR^1(XH) \oplus SHL^5(Q_{19}^{(i)}) \oplus M_3^{(i)}) + (XL \oplus Q_{27}^{(i)} \oplus Q_3^{(i)})$ $H_4^{(i)} = (SHR^3(XH) \oplus Q_{20}^{(i)} \oplus M_4^{(i)}) + (XL \oplus Q_{28}^{(i)} \oplus Q_4^{(i)})$ $H_5^{(i)} = (SHL^6(XH) \oplus SHR^6(Q_{21}^{(i)}) \oplus M_5^{(i)}) + (XL \oplus Q_{29}^{(i)} \oplus Q_5^{(i)})$ $H_6^{(i)} = (SHR^4(XH) \oplus SHL^6(Q_{22}^{(i)}) \oplus M_6^{(i)}) + (XL \oplus Q_{30}^{(i)} \oplus Q_6^{(i)})$ $H_7^{(i)} = (SHR^{11}(XH) \oplus SHL^2(Q_{23}^{(i)}) \oplus M_7^{(i)}) + (XL \oplus Q_{31}^{(i)} \oplus Q_7^{(i)})$ $H_8^{(i)} = ROTL^9(H_4^{(i)}) + (XH \oplus Q_{24}^{(i)} \oplus M_8^{(i)}) + (SHL^8(XL) \oplus Q_{23}^{(i)} \oplus Q_8^{(i)})$ $H_9^{(i)} = ROTL^{10}(H_5^{(i)}) + (XH \oplus Q_{25}^{(i)} \oplus M_9^{(i)}) + (SHR^6(XL) \oplus Q_{16}^{(i)} \oplus Q_9^{(i)})$ $H_{10}^{(i)} = ROTL^{11}(H_6^{(i)}) + (XH \oplus Q_{26}^{(i)} \oplus M_{10}^{(i)}) + (SHL^6(XL) \oplus Q_{17}^{(i)} \oplus Q_{10}^{(i)})$ $H_{11}^{(i)} = ROTL^{12}(H_7^{(i)}) + (XH \oplus Q_{27}^{(i)} \oplus M_{11}^{(i)}) + (SHL^4(XL) \oplus Q_{18}^{(i)} \oplus Q_{11}^{(i)})$ $H_{12}^{(i)} = ROTL^{13}(H_0^{(i)}) + (XH \oplus Q_{28}^{(i)} \oplus M_{12}^{(i)}) + (SHR^3(XL) \oplus Q_{19}^{(i)} \oplus Q_{12}^{(i)})$ $H_{13}^{(i)} = ROTL^{14}(H_1^{(i)}) + (XH \oplus Q_{29}^{(i)} \oplus M_{13}^{(i)}) + (SHR^4(XL) \oplus Q_{20}^{(i)} \oplus Q_{13}^{(i)})$ $H_{14}^{(i)} = ROTL^{15}(H_2^{(i)}) + (XH \oplus Q_{30}^{(i)} \oplus M_{14}^{(i)}) + (SHR^7(XL) \oplus Q_{21}^{(i)} \oplus Q_{14}^{(i)})$ $H_{15}^{(i)} = ROTL^{16}(H_3^{(i)}) + (XH \oplus Q_{31}^{(i)} \oplus M_{15}^{(i)}) + (SHR^2(XL) \oplus Q_{22}^{(i)} \oplus Q_{15}^{(i)})$	

**Table 2.4:** Definition of the folding function  $f_2$  of BLUE MIDNIGHT WISH

placed by each successive intermediate double pipe value (after each message block is processed),  $H^{(i)}$ , and ending with the final double pipe value  $H^{(N)}$ . BMW224 and BMW256 also use two temporary 32-bit words  $XL$  and  $XH$ . The final result of BMW224 is a 224-bit message digest that are actually the least significant 224 bits from the final double pipe, and the final result of BMW256 is a 256-bit message digest that are actually the least significant 256 bits from the final double pipe.

### BMW224 and BMW256 preprocessing

1. Pad the message  $M$ .

2. Parse the padded message into  $N$  512-bit blocks,  $M^{(1)}, M^{(2)}, \dots, M^{(N)}$ .
3. Set the initial double pipe value  $H^{(0)}$  as defined in Table 1.4 for BWM224, or as defined in Table 1.5 for BWM256.

### 2.1.2 BMW384 and BMW512

BMW384 and BMW512 may be used to hash a message  $M$ , having a length of  $l$  bits, where  $0 \leq l < 2^{64}$ . The algorithms use

1. sixteen 64-bit working variables that are part of the double pipe, and
2. additional sixteen 64-bit working variables that together with the variables of the double pipe, make the extended (quadruple) pipe.

The words of the quadruple pipe are labeled  $Q_0^{(i)}, Q_1^{(i)}, \dots, Q_{31}^{(i)}$ . The words of the initial double pipe are labeled  $H_0^{(i)}, H_1^{(i)}, \dots, H_{15}^{(i)}$ , which will hold the initial double pipe value  $H^{(0)}$ , replaced by each successive intermediate double pipe value (after each message block is processed),  $H^{(i)}$ , and ending with the final double pipe value  $H^{(N)}$ . BMW384 and BMW512 also use two temporary 64-bit words  $XL$  and  $XH$ . The final result of BMW384 is a 384-bit message digest that are actually the least significant 384 bits from the final double pipe, and the final result of BMW512 is a 512-bit message digest that are actually the least significant 512 bits from the final double pipe.

#### BMW384 and BMW512 preprocessing

1. Pad the message  $M$ .
2. Parse the padded message into  $N$  1024-bit blocks,  $M^{(1)}, M^{(2)}, \dots, M^{(N)}$ .
3. Set the initial double pipe value  $H^{(0)}$  as defined in Table 1.6 for BWM384, or as defined in Table 1.7 for BWM512.

# Design Rationales

## 3.1 Reasons for default little-endian design

Some of the earlier versions of BLUE MIDNIGHT WISH were designed to be big-endian by default. However, as the designing phase was coming to its end, and we started the optimization phase, we changed the default design to be little-endian since an overwhelming majority of CPU platforms in the world are little-endian.

## 3.2 Reasons for using double pipe iterative structure

In the design of BLUE MIDNIGHT WISH we have decided to incorporate the suggestions of Lucks [2, 3] and Coron et al. [4]. Namely, by setting the size of the chaining pipe to be two times bigger than the hash digest size we are removing weaknesses against the generic attacks of Joux [5] and Kelsey and Schneier [6]. Thus, the double pipe design in BLUE MIDNIGHT WISH guarantees a resistance against generic multicollision attack, and against length extension attacks.

Additionally, as we will see later, using the double pipe concept in combination with the used nonlinear bijections, is a big precaution against differential attacks, because in the differential paths the attacker will have to use two times more variables than in a single pipe (as it is the case in the traditional hashes).

### 3.3 Rationales for used constants in BLUE MIDNIGHT WISH

#### 3.3.1 Constants in logical functions

Logical functions  $s_0, s_1, s_2$  and  $s_3$  are chosen in such a way that they satisfy the following criteria:

- They are bijections in  $\{0, 1\}^{32} \rightarrow \{0, 1\}^{32}$  (resp. in  $\{0, 1\}^{64} \rightarrow \{0, 1\}^{64}$ ).
- They have different pairs of one-bit or two-bits shifts to the left and to the right.
- They have different pairs of rotations to the left, in such a way that one rotation is less than  $w/2, w = 32, 64$ , and the other rotation is bigger than  $w/2$ .
- The values of the rotations that are less than  $w/2$  are in the interval of  $\pm 2$  (resp.  $\pm 4$ ) around numbers  $\{2, 6, 10, 14\}$  (resp.  $\{4, 12, 20, 28\}$ ).
- The values of the rotations that are bigger than  $w/2$  are in the interval of  $\pm 2$  (resp.  $\pm 4$ ) around numbers  $\{18, 22, 26, 30\}$  (resp.  $\{36, 42, 50, 58\}$ ).

By computer search we have found hundreds of such bijections and from them we have chosen four particular  $s_0, s_1, s_2$  and  $s_3$ . The role of these logical functions is to diffuse one-bit difference into 3 or 4 bits difference.

Logical functions  $s_4$  and  $s_5$  are bijections in  $\{0, 1\}^{32} \rightarrow \{0, 1\}^{32}$  (resp. in  $\{0, 1\}^{64} \rightarrow \{0, 1\}^{64}$ ). They have only one shift to the right for just one or two bits. Their role is to spread one-bit differences mostly into two bits (if the difference bit is the right-most or the bit next to the right-most bit, then these functions keep a one-bit difference as a one-bit difference).

Logical functions  $r_1, \dots, r_7$  are rotations with the values that were chosen uniformly at random in the interval  $[1, w - 1]$ .

#### 3.3.2 Constants in the expansion part

In the expansion function  $f_1$  we use the constants  $K_j = j \times (0x05555555), j = 16, 17, \dots, 31$  for BMW224 and BMW256, or the constants  $K_j = j \times (0x0555555555555555), j = 16, 17, \dots, 31$  for BMW384 and BMW512.

The primary reason why we use constants is that we want to avoid the situation that the message  $M = (0, 0, \dots, 0) \equiv \mathbf{0}$  and the double pipe value  $H = (0, 0, \dots, 0) \equiv \mathbf{0}$  are a fixed point. Let

us for a moment omit the upper index  $(i)$  in our notations. If we have in mind that  $(Q_a, Q_b) = f_1(M, f_0(M, H))$ , then if  $f_1$  does not have constants we will have the situation that

$$(\mathbf{0}, \mathbf{0}) = f_1(\mathbf{0}, f_0(\mathbf{0}, \mathbf{0})).$$

We have chosen  $0x05555555$  and  $0x0555555555555555$  as a basis for obtaining 16 constants in the expansion function because we find that their binary presentation as a sequence of alternating 0s and 1s is good source of variety.

The reasons why we choose  $0x05555555$  instead of  $0x55555555$  is simply to avoid complains (warnings) of some C compilers that are finding that  $16 \times (0x55555555)$  is a constant that goes out of the range of a 32-bit word (the reasons are similar for  $0x0555555555555555$ ).

### 3.4 Rationales for the bijective “Step 1” in the function $f_0$

The Step 1 in the definition of the function  $f_0$  is a bijective one when either  $H^{(i-1)}$  or  $M^{(i)}$  are kept constant. Namely the transformation can be expressed as:

$$Q_a = \mathbf{A}_1 \cdot (M^{(i)} \oplus H^{(i-1)}),$$

where we denote  $Q_a = (Q_0^{(i)}, Q_1^{(i)}, \dots, Q_{15}^{(i)})$  and the matrix  $\mathbf{A}_1$  is a  $16 \times 16$  nonsingular matrix in  $\mathbb{Z}_{2^{32}}$  and in  $\mathbb{Z}_{2^{64}}$ . The value of  $\mathbf{A}_1$  is

$$\mathbf{A}_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 1 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 1 \\ -1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 1 & 0 & 0 & -1 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

The matrix  $\mathbf{A}_1$  was obtained from the matrix

$$\mathbf{A}'_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix},$$

by randomly turning some of the values '1' into '-1'. Note that the product  $\mathbf{A}'_1 \cdot M^{(i)}$  can be expressed as:

$$\mathbf{A}'_1 = \text{ROTR}^2(M^{(i)}) + \text{ROTR}^3(M^{(i)}) + \text{ROTR}^6(M^{(i)}) + \text{ROTR}^9(M^{(i)}) + \text{ROTR}^{11}(M^{(i)}),$$

where the operations  $\text{ROTR}^j(M^{(i)})$  are rotations to the right of the vector  $M^{(i)} = (M_0^{(i)}, M_1^{(i)}, \dots, M_{15}^{(i)})$  by  $j$  words and "+" means componentwise addition in  $\mathbb{Z}_{2^{32}}$  (resp. in  $\mathbb{Z}_{2^{64}}$ ). In other words we have that:

$$\begin{aligned} \text{ROTR}^2(M^{(i)}) &= (M_{14}^{(i)}, M_{15}^{(i)}, M_0^{(i)}, \dots, M_{13}^{(i)}) \\ \text{ROTR}^3(M^{(i)}) &= (M_{13}^{(i)}, M_{14}^{(i)}, M_{15}^{(i)}, \dots, M_{12}^{(i)}) \\ \text{ROTR}^6(M^{(i)}) &= (M_{10}^{(i)}, M_{11}^{(i)}, M_{12}^{(i)}, \dots, M_9^{(i)}), \\ \text{ROTR}^9(M^{(i)}) &= (M_7^{(i)}, M_8^{(i)}, M_9^{(i)}, \dots, M_6^{(i)}) \\ \text{ROTR}^{11}(M^{(i)}) &= (M_5^{(i)}, M_6^{(i)}, M_7^{(i)}, \dots, M_4^{(i)}) \end{aligned}$$

and

$$\mathbf{A}'_1 \cdot M^{(i)} = (M_{14}^{(i)} + M_{13}^{(i)} + M_{10}^{(i)} + M_7^{(i)} + M_5^{(i)}, \dots, M_{13}^{(i)} + M_{12}^{(i)} + M_9^{(i)} + M_6^{(i)} + M_4^{(i)}).$$

It is straightforward to prove the following

**Lemma 1.** *The transformation  $\mathbf{A}'_1 \cdot M^{(i)}$  diffuses every one bit difference in the vector  $M^{(i)}$  into at least five bits difference.*  $\square$

The matrix  $\mathbf{A}_1$  is obtained from the matrix  $\mathbf{A}'_1$  by randomly selecting some of the values "1" and turning them into "-1". It is straightforward to prove the following

**Lemma 2.** *The transformation  $\mathbf{A}_1 \cdot M^{(i)}$  diffuses every one bit difference in the vector  $M^{(i)}$  into at least five bits difference.*  $\square$

The reason why we decided to use the transformation  $\mathbf{A}_1 \cdot M^{(i)}$  instead of the transformation  $\mathbf{A}'_1 \cdot M^{(i)}$  is the fact that in any CPU, the computational costs of addition and subtraction are the same, but the component with mixed usage of additions and subtractions is more complex, and it is reasonable to expect the increased complexity to increase the ability to resist cryptanalysis.

### 3.5 Rationales for the bijective “Step 2” in the function $f_0$

The Step 2 in the definition of the function  $f_1$  is also a bijective one, but now the bijective transformation is achieved for every component of the vector  $M^{(i)}$  by applying transformations  $s_0, s_1, s_2, s_3$  and  $s_4$  (see the Table 1.3).

It is easy to prove the following

**Lemma 3.** *The transformations  $s_i, i = 0, \dots, 5$  and  $r_i, i = 1, \dots, 7$  defined in the Table 1.3 are bijective transformations in  $\{0, 1\}^{32}$  (resp. in  $\{0, 1\}^{64}$ ).*

For our analysis of the hash function we denote this bijective Step 2 transformation as  $\mathbf{A}_2 : \{0, 1\}^{16w} \rightarrow \{0, 1\}^{16w}$ . From the composition of Step 1 and Step 2 in the function  $f_0$  it is clear that

$$f_0(M_i, H_{i-1}) \equiv \mathbf{A}_2(\mathbf{A}_1 \cdot (M_i \oplus H_{i-1})).$$

The differential (diffusion) property for  $s_i, i = 0, \dots, 3$  transformations is summarized in the following

**Lemma 4.** *The transformations  $s_0, s_1, s_2$  and  $s_3$  defined in the Table 1.3 diffuse every one bit difference in their arguments (32-bit or 64-bit words) into 3 or 4 bits of difference.  $\square$*

The differential (diffusion) property for  $s_4$  and  $s_5$  transformations is summarized in the following

**Lemma 5.** *The transformations  $s_4$  and  $s_5$  defined in the Table 1.3 diffuse every one bit difference in their arguments (32-bit or 64-bit words) into 1 or 2 bits of difference.  $\square$*

The differential (diffusion) property of consecutive application of Step 1 and Step 2 in the function  $f_0$  can be determined by combining Lemma 4 and Lemma 5 and is summarized in the following

**Lemma 6.** *Every one bit difference in the vector  $M^{(i)}$  or in the vector  $H^{(i-1)}$  after Step 1 and Step 2 of the function  $f_0$  diffuses into 5 words of the the vector  $Q_a$ , and the differences in those 5 words are either 1 or 2 bits difference, or 3 or 4 bits difference.*

### 3.6 Tunable parameters $ExpandRounds_1$ and $ExpandRounds_2$

The goal in the design of  $f_1$  function was with a given  $M^{(i)}$ , in a bijective (block cipher) manner to map the values  $Q_a = (Q_0^{(i)}, Q_1^{(i)}, \dots, Q_{15}^{(i)})$  to the values  $Q_b = (Q_{16}^{(i)}, Q_{17}^{(i)}, \dots, Q_{31}^{(i)})$ . We are achieving that in 16 expansion steps using two types of expansion functions. The first expansion function  $expand_1()$  is more complex and is used in the beginning of the expansion process. In that function, a difference of a one bit in  $M^{(i)}$  or in  $Q_a$  diffuses much faster than in the second expansion function  $expand_2()$ . The numbers of how many times we will call the first and the second function are connected with the following relation:

$$ExpandRounds_1 + ExpandRounds_2 = 16.$$

The function

$$\begin{aligned} expand_1(j) = & s_1(Q_{j-16}^{(i)}) + s_2(Q_{j-15}^{(i)}) + s_3(Q_{j-14}^{(i)}) + s_0(Q_{j-13}^{(i)}) \\ & + s_1(Q_{j-12}^{(i)}) + s_2(Q_{j-11}^{(i)}) + s_3(Q_{j-10}^{(i)}) + s_0(Q_{j-9}^{(i)}) \\ & + s_1(Q_{j-8}^{(i)}) + s_2(Q_{j-7}^{(i)}) + s_3(Q_{j-6}^{(i)}) + s_0(Q_{j-5}^{(i)}) , \\ & + s_1(Q_{j-4}^{(i)}) + s_2(Q_{j-3}^{(i)}) + s_3(Q_{j-2}^{(i)}) + s_0(Q_{j-1}^{(i)}) \\ & + M_{(j-16) \bmod 16}^{(i)} + M_{(j-13) \bmod 16}^{(i)} - M_{(j-6) \bmod 16}^{(i)} + K_j \end{aligned}$$

is more complex and more computationally expensive function in the expansion part. However, as a sort of security/performance tradeoff for the computation of the expanded values, we are using the second simplified expand function:

$$\begin{aligned} expand_2(j) = & Q_{j-16}^{(i)} + r_1(Q_{j-15}^{(i)}) + Q_{j-14}^{(i)} + r_2(Q_{j-13}^{(i)}) \\ & + Q_{j-12}^{(i)} + r_3(Q_{j-11}^{(i)}) + Q_{j-10}^{(i)} + r_4(Q_{j-9}^{(i)}) \\ & + Q_{j-8}^{(i)} + r_5(Q_{j-7}^{(i)}) + Q_{j-6}^{(i)} + r_6(Q_{j-5}^{(i)}) . \\ & + Q_{j-4}^{(i)} + r_7(Q_{j-3}^{(i)}) + s_5(Q_{j-2}^{(i)}) + s_4(Q_{j-1}^{(i)}) \\ & + M_{(j-16) \bmod 16}^{(i)} + M_{(j-13) \bmod 16}^{(i)} - M_{(j-6) \bmod 16}^{(i)} + K_j \end{aligned}$$

Our recommendation for these tunable parameters is:  $ExpandRounds_1 = 2$ ,  $ExpandRounds_2 = 14$ .

#### 3.6.1 Statements, according to the NIST requirements 2.B.1.

Here we give statements, according to the NIST requirements 2.B.1.

##### I.

The following statements are the same for each digest size  $n = 224, 256, 384, 512$ .

**II.**

Using two consecutive  $expand_1()$  rounds at the beginning of the cipher  $f_1$  means that the variables  $Q_a = (Q_0, \dots, Q_{15})$  enter the 16-round block cipher  $f_1$  in two different linear combinations of their bits consecutively (excluding  $Q_0$ , which enters the cipher  $f_1$  directly only once as  $s_1(Q_0)$  and indirectly in  $Q_{17}, \dots, Q_{31}$ ). For instance  $Q_1$  enters  $f_1$  in the first two rounds directly as  $s_2(Q_1)$  and  $s_1(Q_1)$ ,  $Q_2$  enters  $f_1$  in the first two rounds directly as  $s_3(Q_2)$  and  $s_2(Q_2)$ , etc. The more rounds  $expand_1()$  is used, the more linear combinations of variables of  $Q_a$  enter the cipher  $f_1$ .

**III.**

By using more rounds  $expand_1()$  we can increase the strength (and the complexity) of the cipher  $f_1$ , and thus the security of BLUE MIDNIGHT WISH, but we will decrease the speed.

**IV.**

By using two different round functions  $expand_1()$  and  $expand_2()$  we increase the difficulty of finding overall differential paths, because the differentials for the first function  $expand_1()$  and for the second function  $expand_2()$  are completely different.

**V.**

We are not aware of any weaknesses even for  $ExpandRounds_1 = 0$  and  $ExpandRounds_2 = 16$  or  $ExpandRounds_1 = 16$  and  $ExpandRounds_2 = 0$  or any other combination for  $ExpandRounds_1 + ExpandRounds_2 = 16$ , but we propose  $ExpandRounds_1 = 2$  as an optimal tradeoff between security and efficiency.

## 3.7 Cryptanalysis of BLUE MIDNIGHT WISH

### 3.7.1 Bijective parts in the compression function of BLUE MIDNIGHT WISH

Here we will write the compression function in such a way that we will emphasize all its functional entities. Later on, this representation will help us to perform a cryptanalysis of the compression function.

First let us adopt the following notation for this and the next section:

1. We omit the upper index  $(i)$  in our notations.
2. We denote the  $i$ -th message block as  $M_i$  (instead of  $M^{(i)}$ ).
3. We denote the  $(i - 1)$ -th double pipe as  $H_{i-1}$  (instead of  $H^{(i-1)}$ ).
4. We denote the final output from the function  $f_2$  as  $H_i$  i.e.  $H_i = f_2(M_i, Q_a, Q_b)$  (instead of  $H^{(i)}$ ).

Having in mind the definition of the function  $f_2$  given in Table 2.4 we can rewrite the function  $f_2$  as follows.

Let  $f_3 : \{0, 1\}^{2m} \rightarrow \{0, 1\}^m$  is defined as:

$$f_3(M_i, Q_b) = \begin{pmatrix} SHL^5(XH) \oplus SHR^5(Q_{16}^{(i)}) \oplus M_0^{(i)} \\ SHR^7(XH) \oplus SHL^8(Q_{17}^{(i)}) \oplus M_1^{(i)} \\ SHR^5(XH) \oplus SHL^5(Q_{18}^{(i)}) \oplus M_2^{(i)} \\ SHR^1(XH) \oplus SHL^5(Q_{19}^{(i)}) \oplus M_3^{(i)} \\ SHR^3(XH) \oplus Q_{20}^{(i)} \oplus M_4^{(i)} \\ SHL^6(XH) \oplus SHR^6(Q_{21}^{(i)}) \oplus M_5^{(i)} \\ SHR^4(XH) \oplus SHL^6(Q_{22}^{(i)}) \oplus M_6^{(i)} \\ SHR^{11}(XH) \oplus SHL^2(Q_{23}^{(i)}) \oplus M_7^{(i)} \\ XH \oplus Q_{24}^{(i)} \oplus M_8^{(i)} \\ XH \oplus Q_{25}^{(i)} \oplus M_9^{(i)} \\ XH \oplus Q_{26}^{(i)} \oplus M_{10}^{(i)} \\ XH \oplus Q_{27}^{(i)} \oplus M_{11}^{(i)} \\ XH \oplus Q_{28}^{(i)} \oplus M_{12}^{(i)} \\ XH \oplus Q_{29}^{(i)} \oplus M_{13}^{(i)} \\ XH \oplus Q_{30}^{(i)} \oplus M_{14}^{(i)} \\ XH \oplus Q_{31}^{(i)} \oplus M_{15}^{(i)} \end{pmatrix}$$

Further on, let  $f_4 : \{0, 1\}^{2m} \rightarrow \{0, 1\}^m$  is defined as:

$$f_4(Q_a, Q_b) = \begin{pmatrix} XL \oplus Q_{24}^{(i)} \oplus Q_0^{(i)} \\ XL \oplus Q_{25}^{(i)} \oplus Q_1^{(i)} \\ XL \oplus Q_{26}^{(i)} \oplus Q_2^{(i)} \\ XL \oplus Q_{27}^{(i)} \oplus Q_3^{(i)} \\ XL \oplus Q_{28}^{(i)} \oplus Q_4^{(i)} \\ XL \oplus Q_{29}^{(i)} \oplus Q_5^{(i)} \\ XL \oplus Q_{30}^{(i)} \oplus Q_6^{(i)} \\ XL \oplus Q_{31}^{(i)} \oplus Q_7^{(i)} \\ SHL^8(XL) \oplus Q_{23}^{(i)} \oplus Q_8^{(i)} \\ SHR^6(XL) \oplus Q_{16}^{(i)} \oplus Q_9^{(i)} \\ SHL^6(XL) \oplus Q_{17}^{(i)} \oplus Q_{10}^{(i)} \\ SHL^4(XL) \oplus Q_{18}^{(i)} \oplus Q_{11}^{(i)} \\ SHR^3(XL) \oplus Q_{19}^{(i)} \oplus Q_{12}^{(i)} \\ SHR^4(XL) \oplus Q_{20}^{(i)} \oplus Q_{13}^{(i)} \\ SHR^7(XL) \oplus Q_{21}^{(i)} \oplus Q_{14}^{(i)} \\ SHR^2(XL) \oplus Q_{22}^{(i)} \oplus Q_{15}^{(i)} \end{pmatrix}$$

Finally for any  $X = (X_0, X_1, \dots, X_{15})$  where  $X_i$  are  $w$ -bit words ( $w = 32, 64$ ), let us define the function  $f_5 : \{0, 1\}^{16w} \rightarrow \{0, 1\}^{16w}$  as:

$$f_5(X) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ ROTL^9(X_4) \\ ROTL^{10}(X_5) \\ ROTL^{11}(X_6) \\ ROTL^{12}(X_7) \\ ROTL^{13}(X_0) \\ ROTL^{14}(X_1) \\ ROTL^{15}(X_2) \\ ROTL^{16}(X_3) \end{pmatrix}$$

Now the final output from the  $f_2$  function is  $H_i = (H_0, H_1, \dots, H_{15})$  and can be rewritten as:

$$H_i = f_2(M_i, Q_a, Q_b) \equiv f_3(M_i, Q_b) + f_4(Q_a, Q_b) + f_5(f_3(M_i, Q_b) + f_4(Q_a, Q_b)). \quad (3.7.1)$$

One of the basic security properties of BLUE MIDNIGHT WISH is its nonlinear folding function  $f_2$ . We describe here one specially designed part of this function.

Let us denote by  $L_a$  the restriction of the function  $f_3(M_i, Q_b)$  when  $M_i$  is kept constant, i.e.

$$L_a(Q_b) \equiv f_3(\mathbf{Const}_1, Q_b) = \begin{pmatrix} SHL^5(XH) \oplus SHR^5(Q_{16}^{(i)}) \\ SHR^7(XH) \oplus SHL^8(Q_{17}^{(i)}) \\ SHR^5(XH) \oplus SHL^5(Q_{18}^{(i)}) \\ SHR^1(XH) \oplus SHL^5(Q_{19}^{(i)}) \\ SHR^3(XH) \oplus Q_{20}^{(i)} \\ SHL^6(XH) \oplus SHR^6(Q_{21}^{(i)}) \\ SHR^4(XH) \oplus SHL^6(Q_{22}^{(i)}) \\ SHR^{11}(XH) \oplus SHL^2(Q_{23}^{(i)}) \\ XH \oplus Q_{24}^{(i)} \\ XH \oplus Q_{25}^{(i)} \\ XH \oplus Q_{26}^{(i)} \\ XH \oplus Q_{27}^{(i)} \\ XH \oplus Q_{28}^{(i)} \\ XH \oplus Q_{29}^{(i)} \\ XH \oplus Q_{30}^{(i)} \\ XH \oplus Q_{31}^{(i)} \end{pmatrix} \oplus \mathbf{Const}_1$$

Further on, let us denote by  $L_b$  the restriction of the function  $f_4(Q_a, Q_b)$  when  $Q_a$  is kept constant,

i.e.

$$L_b(Qb) \equiv f_4(\mathbf{Const}_2, Qb) = \left( \begin{array}{l} XL \oplus Q_{24}^{(i)} \\ XL \oplus Q_{25}^{(i)} \\ XL \oplus Q_{26}^{(i)} \\ XL \oplus Q_{27}^{(i)} \\ XL \oplus Q_{28}^{(i)} \\ XL \oplus Q_{29}^{(i)} \\ XL \oplus Q_{30}^{(i)} \\ XL \oplus Q_{31}^{(i)} \\ SHL^8(XL) \oplus Q_{23}^{(i)} \\ SHR^6(XL) \oplus Q_{16}^{(i)} \\ SHL^6(XL) \oplus Q_{17}^{(i)} \\ SHL^4(XL) \oplus Q_{18}^{(i)} \\ SHR^3(XL) \oplus Q_{19}^{(i)} \\ SHR^4(XL) \oplus Q_{20}^{(i)} \\ SHR^7(XL) \oplus Q_{21}^{(i)} \\ SHR^2(XL) \oplus Q_{22}^{(i)} \end{array} \right) \oplus \mathbf{Const}_2$$

Finally, let us define the transformation  $L : \{0,1\}^{16w} \rightarrow \{0,1\}^{16w}$  as  $L \equiv L_a \oplus L_b$  i.e.:

$$L(Qb) = \left( \begin{array}{l} XL \oplus Q_{24}^{(i)} \\ XL \oplus Q_{25}^{(i)} \\ XL \oplus Q_{26}^{(i)} \\ XL \oplus Q_{27}^{(i)} \\ XL \oplus Q_{28}^{(i)} \\ XL \oplus Q_{29}^{(i)} \\ XL \oplus Q_{30}^{(i)} \\ XL \oplus Q_{31}^{(i)} \\ SHL^8(XL) \oplus Q_{23}^{(i)} \\ SHR^6(XL) \oplus Q_{16}^{(i)} \\ SHL^6(XL) \oplus Q_{17}^{(i)} \\ SHL^4(XL) \oplus Q_{18}^{(i)} \\ SHR^3(XL) \oplus Q_{19}^{(i)} \\ SHR^4(XL) \oplus Q_{20}^{(i)} \\ SHR^7(XL) \oplus Q_{21}^{(i)} \\ SHR^2(XL) \oplus Q_{22}^{(i)} \end{array} \right) \oplus \left( \begin{array}{l} XL \oplus Q_{24}^{(i)} \\ XL \oplus Q_{25}^{(i)} \\ XL \oplus Q_{26}^{(i)} \\ XL \oplus Q_{27}^{(i)} \\ XL \oplus Q_{28}^{(i)} \\ XL \oplus Q_{29}^{(i)} \\ XL \oplus Q_{30}^{(i)} \\ XL \oplus Q_{31}^{(i)} \\ SHL^8(XL) \oplus Q_{23}^{(i)} \\ SHR^6(XL) \oplus Q_{16}^{(i)} \\ SHL^6(XL) \oplus Q_{17}^{(i)} \\ SHL^4(XL) \oplus Q_{18}^{(i)} \\ SHR^3(XL) \oplus Q_{19}^{(i)} \\ SHR^4(XL) \oplus Q_{20}^{(i)} \\ SHR^7(XL) \oplus Q_{21}^{(i)} \\ SHR^2(XL) \oplus Q_{22}^{(i)} \end{array} \right) \oplus \mathbf{Const}_3$$

**Theorem 1.** *The transformation  $L : \{0,1\}^{16w} \rightarrow \{0,1\}^{16w}$  is bijection for both values  $w = 32$  and  $w = 64$ .*

*Proof.* Direct linear algebra check of the determinant of the corresponding matrix for the transformation  $L$  for both cases  $w = 32$  and  $w = 64$  shows that the determinant is 1 (in  $GF(2)$ ).  $\square$

The constants for shifting left or right used in the transformation  $L$  were found by computer search, such that  $L$  is bijective transformation both for  $w = 32$  and  $w = 64$ .

The following theorem is true about different bijective parts of the compression function of BLUE MIDNIGHT WISH :

**Theorem 2.**

1. When  $H_{i-1}$  is fixed,  $f_0(M_i, H_{i-1})$  is a bijection.
2. When  $M_i$  is fixed,  $f_0(M_i, H_{i-1})$  is a bijection.
3. When  $Q_a$  is fixed,  $f_1(M_i, Q_a)$  is a bijection.
4. When  $M_i$  is fixed,  $f_1(M_i, Q_a)$  is a bijection.
5. When  $Q_b$  and  $M_i$  are fixed,  $f_2(M_i, Q_a, Q_b)$  is a bijection.
6. When  $Q_b$  and  $Q_a$  are fixed,  $f_2(M_i, Q_a, Q_b)$  is a bijection.

*Proof.* **Item 1.** This is actually rephrasing of the title of the Section 3.4 and is a consequence of the non-singularity of the matrix  $\mathbf{A}_1$ .

**Item2.** This is rephrasing of the title of the Section 3.5 and the Lemma 3.

**Item 3.** (sketch) Let us take  $Q_b = f_1(M_i, Q_a)$ , where the values of  $Q_a$  are given and fixed. Then, for every given value of  $Q_b$  we can obtain the following equation:

$$\mathbf{B} \cdot M = \mathbf{Const}$$

where  $\mathbf{Const} = g(Q_a, Q_b)$  is some function obtained from the expanding functions  $expand_1()$  and  $expand_2()$  and where the matrix

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

is a nonsingular matrix in the ring  $(\mathbb{Z}_{2^{32}}, +, \times)$  and in the ring  $(\mathbb{Z}_{2^{64}}, +, \times)$ .

**Item 4.** Let the value of  $M_i = (M_0, \dots, M_{15})$  is given and fixed. Then, for every given value

$Q_b = (Q_{16}, \dots, Q_{31})$  in a unique way we can obtain the values

$$\begin{aligned} Q_{15} &= \text{expand}_2^{(-1)}(31) \\ Q_{14} &= \text{expand}_2^{(-1)}(30) \\ &\dots \\ Q_2 &= \text{expand}_2^{(-1)}(18) \\ Q_1 &= \text{expand}_1^{(-1)}(17) \\ Q_0 &= \text{expand}_1^{(-1)}(16) \end{aligned}$$

where

$$\begin{aligned} \text{expand}_2^{(-1)}(j) = & \quad Q_j \quad - \quad r_1(Q_{j-15}) \quad - \quad Q_{j-14} \quad - \quad r_2(Q_{j-13}) \\ & - \quad Q_{j-12} \quad - \quad r_3(Q_{j-11}) \quad - \quad Q_{j-10} \quad - \quad r_4(Q_{j-9}) \\ & - \quad Q_{j-8} \quad - \quad r_5(Q_{j-7}) \quad - \quad Q_{j-6} \quad - \quad r_6(Q_{j-5}) \quad . \\ & - \quad Q_{j-4} \quad - \quad r_7(Q_{j-3}) \quad - \quad s_5(Q_{j-2}) \quad - \quad s_4(Q_{j-1}) \\ & - M_{(j-16) \bmod 16}^{(i)} - M_{(j-13) \bmod 16}^{(i)} + M_{(j-6) \bmod 16}^{(i)} - K_j \end{aligned}$$

and

$$\begin{aligned} \text{expand}_1^{(-1)}(j) = & \quad s_1^{-1} \left( Q_j \quad - \quad s_2(Q_{j-15}) \quad - \quad s_3(Q_{j-14}) \quad - \quad s_0(Q_{j-13}) \right. \\ & - \quad s_1(Q_{j-12}) \quad - \quad s_2(Q_{j-11}) \quad - \quad s_3(Q_{j-10}) \quad - \quad s_0(Q_{j-9}) \\ & - \quad s_1(Q_{j-8}) \quad - \quad s_2(Q_{j-7}) \quad - \quad s_3(Q_{j-6}) \quad - \quad s_0(Q_{j-5}) \quad , \\ & - \quad s_1(Q_{j-4}) \quad - \quad s_2(Q_{j-3}) \quad - \quad s_3(Q_{j-2}) \quad - \quad s_0(Q_{j-1}) \\ & \left. - M_{(j-16) \bmod 16}^{(i)} - M_{(j-13) \bmod 16}^{(i)} + M_{(j-6) \bmod 16}^{(i)} - K_j \right) \end{aligned}$$

**Item 5.** (sketch) If  $Q_b$  and  $M_i$  are fixed then  $H_i = f_2(M_i, Q_a, Q_b)$  can be rewritten as

$$H_i = (L_a(Q_b) \oplus M_i) + (L_b(Q_b) \oplus Q_a) = \mathbf{Const}_1(Q_b, M_i) + (\mathbf{Const}_2(Q_b, M_i) \oplus Q_a),$$

where  $\mathbf{Const}_1(Q_b, M_i)$  and  $\mathbf{Const}_2(Q_b, M_i)$  are some expressions of the constants  $Q_b$  and  $M_i$ . This is a bijection of  $Q_a$ .

**Item 6.** (sketch) If  $Q_a$  and  $Q_b$  are fixed then  $H_i = f_2(M_i, Q_a, Q_b)$  can be rewritten as

$$H_i = (L_a(Q_b) \oplus M_i) + (L_b(Q_b) \oplus Q_a) = (\mathbf{Const}_1(Q_a, Q_b) \oplus M_i) + \mathbf{Const}_2(Q_a, Q_b),$$

where  $\mathbf{Const}_1(Q_a, Q_b)$  and  $\mathbf{Const}_2(Q_a, Q_b)$  are some expression of the constants  $Q_a$  and  $Q_b$ . This is a bijection of  $M_i$ .

□

### 3.7.2 Representation as a generalized PGV6 scheme

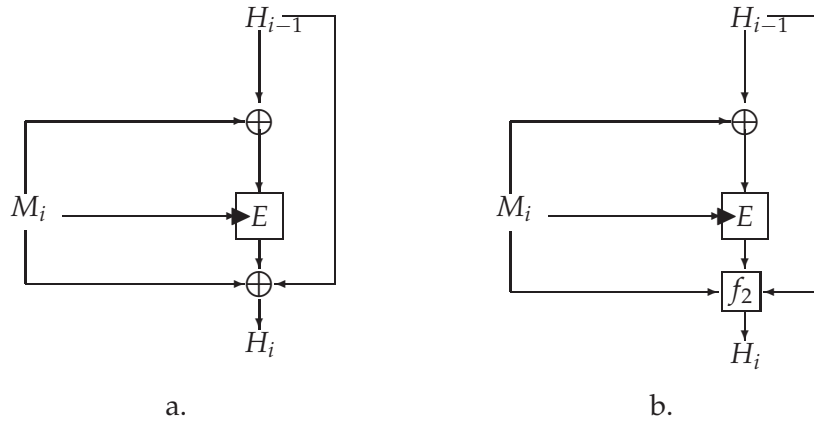
Preneel, Govaerts, and Vandewalle in [7] have located 12 secure schemes for constructing hash functions from block ciphers. Black et. al., [8] have proved (in ideal cipher model) that those schemes are collision-resistant too.

The basic iterative relation for the scheme number 6 (PGV6) is:

$$H_i = E(M_i, M_i \oplus H_{i-1}) \oplus M_i \oplus H_{i-1}$$

where the notation  $E(K, X)$  denotes a block cipher operation with a key  $K$  and a plaintext  $X$ .

The graphical representation of the scheme is given in Figure 3.1a.



**Figure 3.1:** **a.** The PGV6 one-way compression function, **b.** Generalized PGV6 one-way compression function where the feedback information of  $M_i$  and  $H_{i-1}$  is combined with the ciphertext  $E(M_i, M_i \oplus H_{i-1})$  not with simple xor function  $\oplus$  but with some more complex function  $f_2$ .

The feedback information that is used in PGV is the expression  $M_i \oplus H_{i-1}$  and the scheme can be expressed as:  $H_i = f_2(M_i, H_{i-1}, E(M_i, H_{i-1}))$  where

$$f_2(M_i, H_{i-1}, E(M_i, H_{i-1})) \equiv E(M_i, H_{i-1}) \oplus M_i \oplus H_{i-1}.$$

However, we can transform the feedback information with some generalized function  $f_2$  and that generalized PGV6 scheme is shown on Figure 3.1b.

**Theorem 3.** BLUE MIDNIGHT WISH hash function can be expressed as a generalized PGV6 scheme.

*Proof.* First, recall that we have so far represented the the BLUE MIDNIGHT WISH as:  $H_i = f_2(M_i, Q_a, Q_b)$ , where  $Q_a = f_0(M_i, H_{i-1}) = \mathbf{A}_2(\mathbf{A}_1 \cdot (M_i \oplus H_{i-1}))$ , and where  $Q_b = f_1(M_i, Q_a) =$

$f_1(M_i, f_0(M_i, H_{i-1}))$ . So, in the composition of bijections  $f_0$  and  $f_1$  we actually have the xoring part of the PGV6 scheme (the xoring  $M_i \oplus H_{i-1}$ ). In the expression for  $Q_b = f_1(M_i, f_0(M_i, H_{i-1}))$  we can treat  $M_i$  as a key in the block cipher:

$$f_1(M_i, \mathbf{A}_2(\mathbf{A}_1 \cdot (M_i \oplus H_{i-1}))) \equiv E(M_i, M_i \oplus H_{i-1}).$$

By all this, we have three components  $M_i$ ,  $H_{i-1}$ , and  $E(M_i, M_i \oplus H_{i-1})$  that are functionally combined by the function  $f_2$  (instead of the usual combination by the operations  $\oplus$  as it is done in PGV6 scheme) i.e., BLUE MIDNIGHT WISH can be represented as

$$H_i = f_2(M_i, H_{i-1}, E(M_i, M_i \oplus H_{i-1})).$$

□

**Note.** The underlying block cipher  $f_1$  used in BLUE MIDNIGHT WISH is not ideal. As we will show in the following subsection its first word (32-bit or 64-bit) in its output is distinguishable from an ideal random function. However, this deficiency of the used block cipher in BLUE MIDNIGHT WISH is compensated with the more complex feedback function and with the size of the block cipher output (it is two times bigger than the output of the hash function).

### 3.7.3 Representation as a generalized PGV scheme

The discussion in the previous section can be further generalized, and we will show in this section that BLUE MIDNIGHT WISH can be seen as a generalized scheme of any of the 12 PGV secure schemes.

Let us recall the general scheme that authors of PGV paper [7] have considered. They have considered the following iterative scheme for construction of a hash function:

$$H_i = F(H_{i-1}, M_i) \equiv E(a, b) \oplus c,$$

where  $a, b, c \in \{H_{i-1}, M_i, H_{i-1} \oplus M_i, const\}$ , and where  $E(a, b)$  denotes block cipher  $E$  with a key  $a$  and a plaintext  $b$ .

**Theorem 4.** BLUE MIDNIGHT WISH could be seen as a generalization of any of the secure schemes PGV1, PGV2, ... PGV12.

*Proof.* For the purpose of this proof let us denote the key, plaintext, and ciphertext as  $K = M_i$ ,  $PT = M_i \oplus H_{i-1}$ ,  $CT = Q_b = E(M_i, M_i \oplus H_{i-1})$ . Recall that

$$H_i = f_3(M_i, Q_b) + f_4(Q_a, Q_b) + f_5(f_3(M_i, Q_b) + f_4(Q_a, Q_b))$$

is bijectively equivalent to

$$H_i = f_3(M_i, Q_b) + f_4(Q_a, Q_b).$$

We will study the last expression.

$$H_i = (L_a(Q_b) \oplus M_i) + (L_b(Q_b) \oplus Q_a) = (L_a(Q_b) \oplus M_i) + (L_b(Q_b) \oplus f_0(M_i \oplus H_{i-1})).$$

From another point of view we have

$$H_i = (L_a(CT) \oplus K) + (L_b(CT) \oplus f_0(PT)).$$

We know that  $L_a(X) \oplus L_b(X) = L(X)$  is a bijective transformation of  $X$ , and  $f_0$  is a bijective, therefore

1. When we see  $H_i = (L_a(CT) \oplus K) + (L_b(CT) \oplus f_0(PT))$  as a generalization of  $H_i = CT \oplus K \oplus PT$  we obtain the output of the schemes PGV3, PGV4, PGV7, PGV8, PGV11 and PGV12.
2. When we see  $H_i = (L_a(CT) \oplus K) + (L_b(CT) \oplus f_0(PT))$  as a generalization of  $H_i = CT \oplus PT$  we obtain the output of the scheme PGV1, PGV2, PGV5, PGV6, PGV9 and PGV10.

□

The bijective property of  $L$ , is important part in the resistance of BLUE MIDNIGHT WISH against attacks for finding preimages and pseudo-collisions.

We will illustrate the last claim with a representation of a sequence of simplified versions of BLUE MIDNIGHT WISH .

- The original BLUE MIDNIGHT WISH can be represented by the equation (3.7.1) i.e. as

$$H_i = f_3(M_i, Q_b) + f_4(Q_a, Q_b) + f_5(f_3(M_i, Q_b) + f_4(Q_a, Q_b)).$$

- We would get a simpler version of the hash function if we remove the function  $f_5$ . In that case the iterative equation would be

$$H_i = f_3(M_i, Q_b) + f_4(Q_a, Q_b).$$

- Even simpler version of the hash function can be obtained if we change the operation "+" (the additions modulo  $2^{32}$  or modulo  $2^{64}$ ) with the operation  $\oplus$  (bitwise xoring of 32-bit or 64-bit words). In that case the iterative equation would be

$$H_i = f_3(M_i, Q_b) \oplus f_4(Q_a, Q_b).$$

The last equation can be rewritten as:

$$H_i = L(Q_b) \oplus M_i \oplus Q_a.$$

- We can simplify the last iterative equation even further by replacing the values of  $Q_a = f_0(M_i, H_{i-1})$  with the values of  $H_{i-1}$ . In that case we obtain the following simplified BLUE MIDNIGHT WISH hash function:

$$H_i = L(Q_b) \oplus M_i \oplus H_{i-1}. \quad (3.7.2)$$

If we recall that  $Q_b$  is the "ciphertext" i.e. the result of our block cipher  $f_1$ , that encrypts the "plaintext"  $M_i \oplus H_{i-1}$ , with the key  $M_i$ , and  $H_{i-1}$  being previous hash value, we actually have the PGV6 construction, with the exception that instead of direct use of the ciphertext  $Q_b$  we are using some bijective transformation of  $Q_b$  i.e. we are using  $L(Q_b)$ .

A pseudo-collision for the last simplified hash function represented by the equation (3.7.2) is a situation when we have two pairs  $(M'_i, H'_{i-1})$  and  $(M''_i, H''_{i-1})$  such that  $H'_i = H''_i$  where  $H'_i = L(Q'_b) \oplus M'_i \oplus H'_{i-1}$  and  $H''_i = L(Q''_b) \oplus M''_i \oplus H''_{i-1}$  and where  $Q'_b = f_1(M'_i, H'_{i-1})$ ,  $Q''_b = f_1(M''_i, H''_{i-1})$ .

Although we can not directly use the provisions from the PGV6 construction since our block cipher  $f_1$  is not acting as an ideal block cipher, having in mind the complex binary transformation of the "ciphertext"  $Q_b$  and the size of the blocks and keys in the block cipher  $f_1$  that are two times bigger than the hash digest  $n$  we can still claim that finding pseudo-collisions for the last simplified version of BLUE MIDNIGHT WISH is infeasible.

### 3.7.4 Monomial tests on the block ciphers used in BLUE MIDNIGHT WISH

The monomial tests have been introduced several years ago by Filiol [9] to evaluate the statistical properties of symmetric ciphers. Later, Saarinen [10] proposed an extension of Filiol's ideas to a chosen IV statistical attack, called the "d-monomial test", and used it to find weaknesses in several proposed stream ciphers. In 2007 Englund, Johansson and Turan [11] generalized Saarinen's idea and proposed a framework for chosen IV statistical attacks using a polynomial description. Their

basic idea is to select a subset of IV bits as variables, assuming all other IV values as well as the key being fixed. Then by obtaining the algebraic normal form for such a function they were searching for some statistical deviations from ideal random Boolean function. A similar approach as that of Englund et al. is also described by O'Neil in [12].

In order to get a statistical measure of the deviation from ideal random Boolean function of the block cipher that is used in BLUE MIDNIGHT WISH we have defined NANT - A Normalized Average Number of Terms (monomials). NANT can be seen as a variant of Englund's monomial tests and it is defined in what follows.

Let  $n \geq r \geq 1$  be integers and let  $F : \{0,1\}^n \rightarrow \{0,1\}^r$  be a vector valued Boolean function. The vector valued function  $F$  can be represented as an  $r$ -tuple of Boolean functions  $F = (F^{(1)}, F^{(2)}, \dots, F^{(r)})$ , where  $F^{(s)} : \{0,1\}^n \rightarrow \{0,1\}$  ( $s = 1, 2, \dots, r$ ), and the value of  $F^{(s)}(x_1, \dots, x_n)$  equals the value of the  $s$ -th component of  $F(x_1, \dots, x_n)$ . The Boolean functions  $F^{(s)}(x_1, \dots, x_n)$  can be expressed in the Algebraic Normal Form (ANF) as polynomials with  $n$  variables  $x_1, \dots, x_n$  of kind  $a_0 \oplus a_1 x_1 \oplus \dots \oplus a_n x_n \oplus a_{1,2} x_1 x_2 \oplus \dots \oplus a_{n-1,n} x_{n-1} x_n \oplus \dots \oplus a_{1,2,\dots,n} x_1 x_2 \dots x_n$ , where  $a_\lambda \in \{0,1\}$ . Each ANF have up to  $2^n$  terms (i.e. monomials), depending of the values of the coefficients  $a_\lambda$ . Denote by  $L_{F^{(s)}}$  the number of terms in the ANF of the function  $F^{(s)}$ . Then the number of terms of the vector valued function  $F$  is defined to be the number  $L_F = \sum_{s=1}^r L_{F^{(s)}}$ .

**Definition 1.** Let  $F : \{0,1\}^n \rightarrow \{0,1\}^r$  be a vector valued Boolean function. For any  $k \in \{1, \dots, n\}$  and any assembly of  $S$  subsets  $\sigma_j = \{i_1, i_2, \dots, i_k\} \subset \{0, 1, \dots, n-1\}$  chosen uniformly at random ( $1 \leq j \leq S$ ), let  $F_{\sigma_j}$  denote the restriction of  $F$  defined by

$$F_{\sigma_j}(x_1, x_2, \dots, x_n) = F(0, \dots, 0, x_{i_1}, 0, \dots, 0, x_{i_2}, 0, \dots, 0, x_{i_k}, 0, \dots, 0).$$

We define a random variable  $\overline{L}_F$  – the Normalized Average Number of Terms (NANT) as:

$$\overline{L}_F = \overline{L}_F(r, k) = \frac{1}{r} \cdot \frac{1}{2^{k-1}} \cdot \lim_{S \rightarrow \infty} \frac{1}{S} \sum_{j=1}^S L_{F_{\sigma_j}}.$$

Since the subsets  $\sigma_j$  are chosen uniformly at random, the average values of  $L_{F_{\sigma_j}^{(s)}}$  ( $s = 1, 2, \dots, r$ ) are  $2^{k-1}$  and the average value of  $L_{F_{\sigma_j}}$  is  $r2^{k-1}$ . Also,  $L_{F_{\sigma_j}^{(s)}} \leq 2^k$ . So, the following theorem is true:

**Theorem 5.** For any function  $F : \{0,1\}^n \rightarrow \{0,1\}^r$  chosen uniformly at random from the set of all such functions, for any value of  $r \geq 1$  and for any  $k \in \{1, \dots, n\}$ , it is true that

$$0 \leq \overline{L}_F \leq 2$$

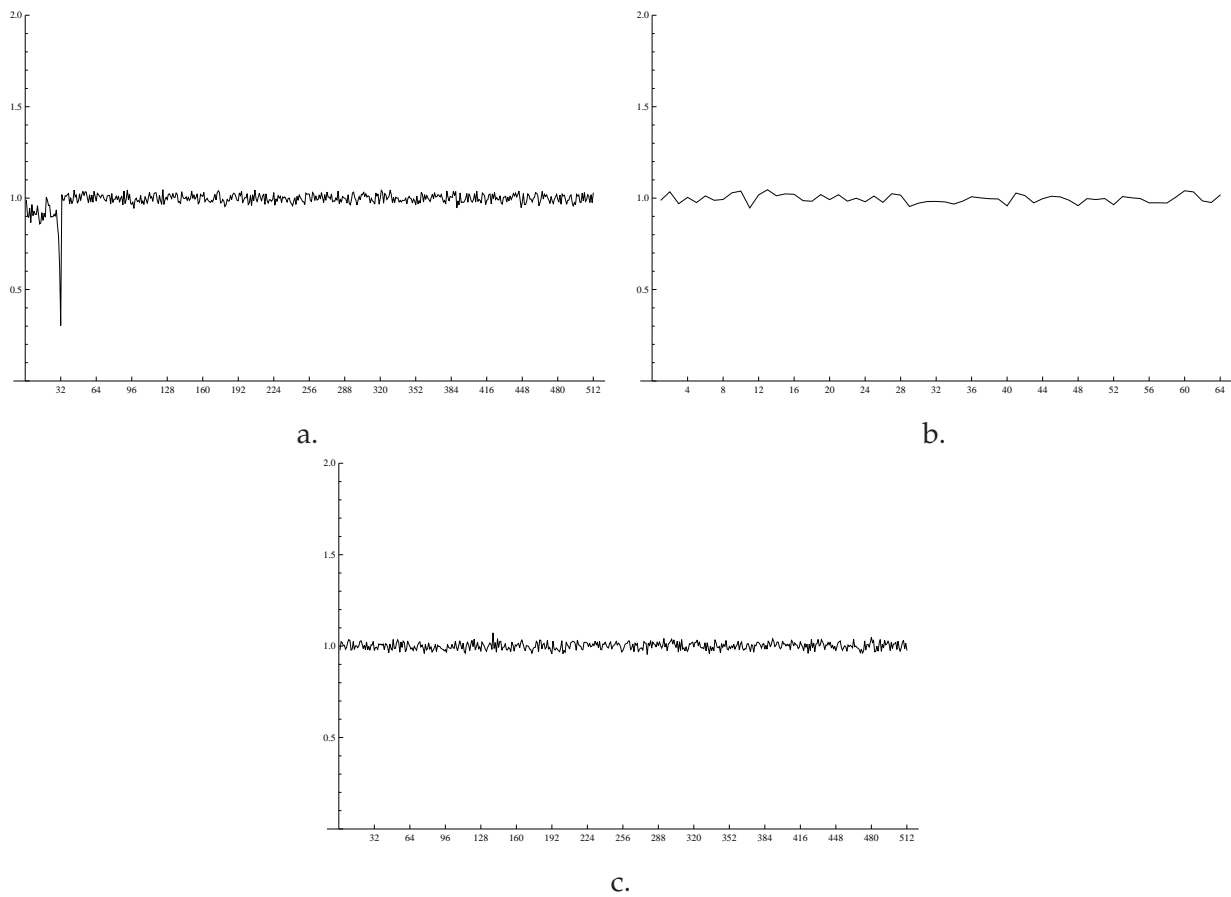
and that the expected value is

$$EX(\overline{L}_F) = 1.$$

□

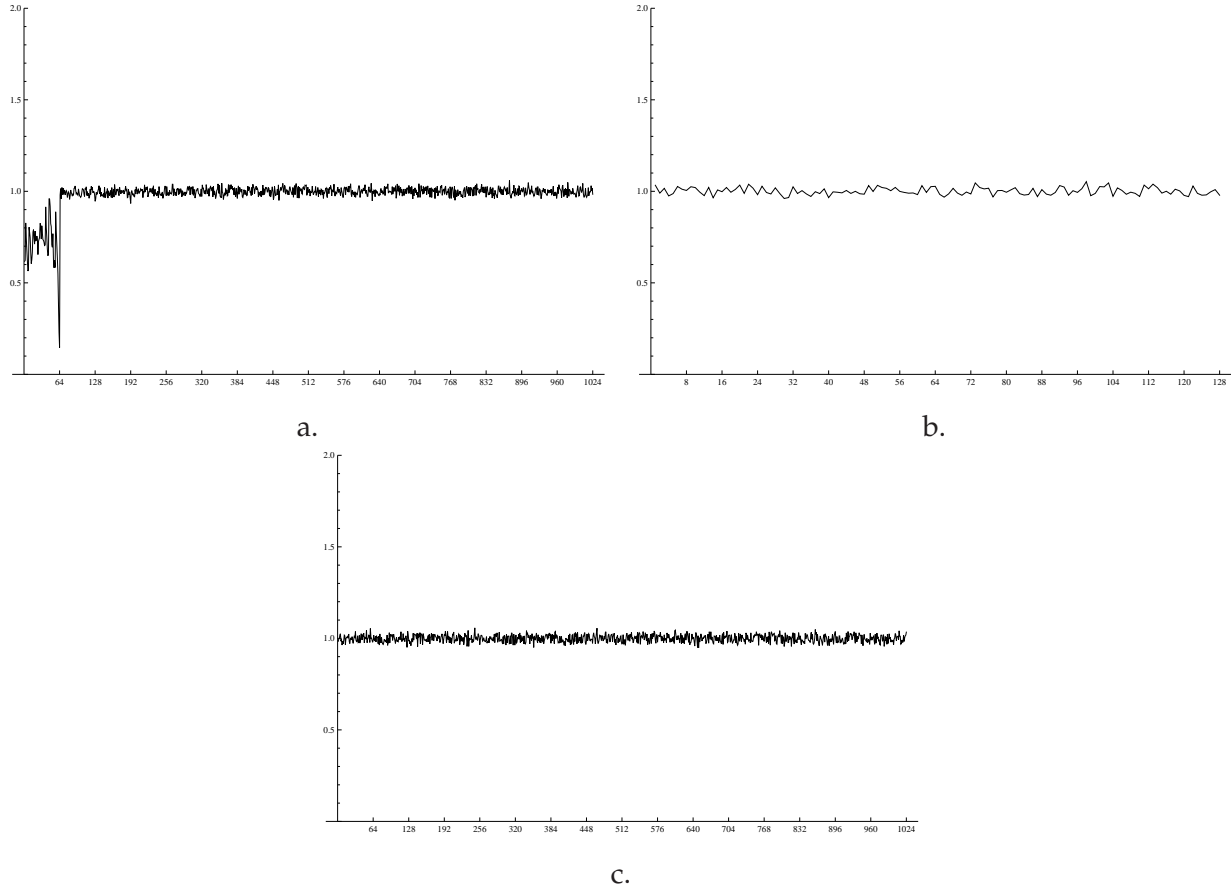
Note that if we want to apply the NANT measure on every bit of some function  $F : \{0,1\}^n \rightarrow \{0,1\}^r$  then instead of averaging on all  $r$  coordinates we are taking that  $r = 1$  i.e., we have to apply the following formula:

$$\overline{L}_F = \overline{L}_F(k) = \frac{1}{2^{k-1}} \cdot \lim_{S \rightarrow \infty} \frac{1}{S} \sum_{j=1}^S L_{F\sigma_j}.$$



**Figure 3.2:** NANT analysis when BMW256 is seen as generalized PGV6 scheme. **a.** Values of  $\overline{L}_F$  for every bit (in total 512 bits) in  $Q_b$ . **b.** Values of  $\overline{L}_F$  for every bit in  $(XL, XH)$  (in total 64 bits). **c.** Values of  $\overline{L}_F$  for every bit (in total 512 bits) in  $H_i$ .

We have measured NANT for every bit of  $Q_b = (Q_{16}, \dots, Q_{31})$ , the pair  $(XL, XH)$  and the final chaining value  $H_i = (H_0, \dots, H_{15})$  by considering BLUE MIDNIGHT WISH as a generalized



**Figure 3.3:** NANT analysis when BMW512 is seen as generalized PGV6 scheme. **a.** Values of  $\overline{L}_F$  for every bit (in total 1024 bits) in  $Q_b$ . **b.** Values of  $\overline{L}_F$  for every bit in  $(XL, XH)$  (in total 128 bits). **c.** Values of  $\overline{L}_F$  for every bit (in total 1024 bits) in  $H_i$ .

PGV6 scheme. In that case, the mapping (the block cipher)  $f_1(M_i, H_{i-1}) = f_1(M_i, f_0(M_i, H_{i-1})) \equiv E(M_i, M_i \oplus H_{i-1})$  was tested with a fixed  $M_i$  in the role of a key.

By performing the NANT tests, we see that the block cipher operation  $E(M_i, M_i \oplus H_{i-1})$  used in BLUE MIDNIGHT WISH is distinguishable from a random permutation. So, when we see BMW256 as a generalized PGV6 scheme, Boolean functions for the bits in  $Q_{16}$  are easily distinguishable from random Boolean function, while for all other variables in  $Q_b$  the Boolean functions for every bit act as a random Boolean function. That is shown in Figure 3.2a. For the two variables  $(XL, XH)$  which consist in total of 64 bits there are no significant deviations from the value 1.0 and that is shown in Figure 3.2b. For the chaining variable  $H_i$  there are also no significant deviations from the value 1.0 (Figure 3.2c).

For digest sizes of 384 and 512 bits we have applied NANT tests on BMW512. The outcome of

the NANT tests is similar with the case of BMW256. Namely, Boolean functions for the bits in  $Q_{16}$  are easily distinguishable from random Boolean function, while for all other variables in  $Q_b$  the Boolean functions for every bit act as a random Boolean function. That is shown in Figure 3.3a. For the two variables ( $XL, XH$ ) which consist in total of 128 bits there are no significant deviations from the value 1.0 and that is shown in Figure 3.3b. For the chaining variable  $H_i$  there are also no significant deviations from the value 1.0 (Figure 3.3c).

So we can say that although BLUE MIDNIGHT WISH follows the well established and secure schemes for designing hash functions from block ciphers (PGV6), its underlying block cipher is weak block cipher. Does it make overall design weak? We think that it does not make overall hash function weak because of the following reasons:

1. The deficiency coming from the distinguishability of the first word (first 4 words) is compensated by the wide block size in BLUE MIDNIGHT WISH which is 512 or 1024 bits long.
2. From the fifth word, all other words in  $Q_b$  are not distinguishable from random 32-bit (64-bit) variables.
3. The feedback information is much more complex function of the initial inputs to the block cipher and its output.

Additionally to the arguments described above, we want to justify our recommendation for the value  $ExpandRounds_1 = 2$ . Namely, from the NANT analysis we have that the variable  $Q_{17}$  which is obtained by the  $expand_1()$  function is already reaching the level of a random Boolean function. So, we can allow the rest of the variables in  $Q_b$  (the variables  $Q_{18}, \dots, Q_{31}$ ) to be computed by the faster and less complex expansion function  $expand_2()$ .

### 3.7.5 Infeasibility of finding collisions, preimages and second preimages

The design of BLUE MIDNIGHT WISH heavily uses combinations of bitwise operations of XORing, rotating and shifting (which can be seen as linear operations in  $GF(2^{32})$  and in  $GF(2^{64})$ ) and operations of addition and subtraction in  $\mathbb{Z}_{2^{32}}$  or in  $\mathbb{Z}_{2^{64}}$  (which are nonlinear operations in  $GF(2^{32})$  and in  $GF(2^{64})$ ). This strategy, combined with the mathematical property of  $f_1$  to be a permutation both when the message block is kept constant or when the previous chaining value is kept constant allows its design to be represented as generalized PGV6 scheme. PGV6 design is second-preimage resistant and collision resistant, and that is the reason why we claim that BLUE MIDNIGHT WISH is also second-preimage resistant and collision resistant hash function.

Additionally, the diffusion characteristics of Boolean functions  $s_i()$ ,  $i = 0, 1, \dots, 5$ , the size of the chaining value being two times wider than the final message digest size, and the nonlinear expressions used in the function  $f_2$ , are the cornerstones of the BLUE MIDNIGHT WISH strength.

More specifically, the chaining part of BLUE MIDNIGHT WISH – “The Double Pipe” is created by the folding function  $f_2$  from three inputs: the current message block itself (xored with the old double pipe), its first bijective transformation  $Q_a$  and its second bijective transformation  $Q_b$ . We can treat  $Q_a$  and  $Q_b$  as ciphertexts, created by non-linear block ciphers, but in a specific manner that they are bijectively tied together. The bijective entanglement, combined with the nonlinearity of the expressions in  $f_2$  gives us confidence that it is infeasible to find collisions, preimages or second preimages of BLUE MIDNIGHT WISH. We believe that it is hard to find a way to change consistently all three inputs (tied by non-linear bijective mappings) in such a way that these changes in 3-times wider input of the compression function  $f_2$  will cancel each other or will lead to controllable changes.

The BLUE MIDNIGHT WISH entanglement of the message, previous double pipe and the next double pipe is given also in the Figure 2.2 for the compression function.

### 3.7.6 Approximation of additions and subtractions with XORs

As mentioned in previous subsection the compression function of BLUE MIDNIGHT WISH uses bitwise operations of XORing, rotating and shifting (as linear operations in  $GF(2^{32})$  and in  $GF(2^{64})$ ) and operations of addition and subtraction in  $\mathbb{Z}_{2^{32}}$  or in  $\mathbb{Z}_{2^{64}}$  (as nonlinear operations in  $GF(2^{32})$  and in  $GF(2^{64})$ ).

A natural idea is to try to find values for which additions and subtractions behave as XORs. In such a case, one would have a completely linear system in the ring  $(\mathbb{Z}_2^n, +, \times)$  for which collisions, preimages and second preimages can easily be found. However, getting all the additions to behave as XORs is a hard.

There are several significant works that are related with analysis of differential probabilities of operations that combine additions modulo  $2^n$ , XORs and left rotations. In 1993 Berson have made a differential cryptanalysis of addition modulo  $2^{32}$  and applied it on MD5 [13], in 2001 Lipmaa and Moriai, have constructed efficient algorithms for computing differential properties of addition modulo  $2^n$  [14], and Lipmaa, Wallén and Dumas in 2004 have constructed linear-time algorithm for computing the additive differential probability of exclusive-or [15].

All of these works are determining the additive differential probability of exclusive-or:

$$Pr[((x + \alpha) \oplus (y + \beta)) - (x \oplus y) = \gamma]$$

and exclusive-or differential probability of addition:

$$Pr[((x \oplus \alpha) + (y \oplus \beta)) \oplus (x + y) = \gamma]$$

where probability is computed for all pairs  $(x, y) \in \mathbb{Z}_{2^n} \times \mathbb{Z}_{2^n}$  and for any predetermined triplet  $(\alpha, \beta, \gamma) \in \mathbb{Z}_{2^n} \times \mathbb{Z}_{2^n} \times \mathbb{Z}_{2^n}$ .

Recently Paul and Preneel [16] have successfully solved the problem of finding solutions in polynomial time of differential equations of addition with two variables  $x$  and  $y$  of type  $(x + y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma$  where  $\alpha, \beta$  and  $\gamma$  are constants. Someone can use their algorithm to try to attack BLUE MIDNIGHT WISH . The problem is that their algorithm is for equations with two variables, and their strategy extended to solving systems of differential equations of addition with three or more variables has exponential complexity i.e. is of the order  $O(2^{b \times k})$  where  $b$  is the bit length of the variables, and  $k$  is the number of equations.

So, in the case of BLUE MIDNIGHT WISH , instead of simple combination of two 32-bit (or 64-bit) variables once by additions then by xoring, we have a complex multivariate system of equations. In these equations both bitwise operations (XORing, shifting or rotation) and word-oriented operations (addition or subtraction) are mutually embedded one into the other. In this moment we do not see how these results will help in finding solutions for these equations.

### 3.7.7 Cryptanalysis of a scaled down BLUE MIDNIGHT WISH

In order to gain a knowledge how robust and sound is the design of BLUE MIDNIGHT WISH , we analyzed a scaled down version of the algorithm. However, down-scaling of BLUE MIDNIGHT WISH required a different approach than that which is usually taken when the hash function has a big number of internal iterative steps.

Namely, BLUE MIDNIGHT WISH does not have iterative steps. It has 16 expansion steps but those steps can not be reduced (since it will destroy the essence of the design - working with a multipermutation). We have decided to down-scale the BLUE MIDNIGHT WISH by reducing the size of the word to 4 bits (corresponding to BMW224 and BMW256) and to 8 bits (corresponding to BMW384 and BMW512). In such a case we defined BMW28 (which has output of 7, 4-bit words i.e. 28 bits), BMW32 (which has output of 8, 4-bit words i.e. 32 bits), BMW48 (which has output of 6, 8-bit

words i.e. 48 bits) and BMW64 (which has output of 8, 8-bit words i.e. 64 bits). The summary is given in Table 3.1.

Algorithm abbreviation	Block size $m$ (in bits)	Word size $w$ (in bits)	Digest size $n$ (in bits)
BMW28	64	4	28
BMW32	64	4	32
BMW48	128	8	48
BMW64	128	8	64

**Table 3.1:** Basic properties of scaled-down variants of the BLUE MIDNIGHT WISH

In order this down-scaling to be correct we had to change (adapt) the used logical functions. In the Table 3.2 we are giving the logical functions that we have used in the down-scaled version of BLUE MIDNIGHT WISH . Note that we use the notation  $ROTL^0(x) \equiv x$  in order to show the consistence of the shape of logical functions in the scale-down function with the original construction of BLUE MIDNIGHT WISH . All logical functions in the scaled-down hash function, similarly as in the original construction are bijections in  $GF(2^w)$  where  $w = 4, 8, 32, 64$ , is the size of the word on which these functions operate on. The initial double-pipe value  $H$  for this scaled-down functions had the value of the  $w$  least significant bits of the double-pipe  $H$  in the original design.

BMW28/BMW32	BMW48/BMW64
$s_0(x) = SHR^1(x) \oplus SHL^1(x) \oplus ROTL^0(x) \oplus ROTL^3(x)$	$s_0(x) = SHR^1(x) \oplus SHL^1(x) \oplus ROTL^3(x) \oplus ROTL^4(x)$
$s_1(x) = SHR^1(x) \oplus SHL^2(x) \oplus ROTL^1(x) \oplus ROTL^3(x)$	$s_1(x) = SHR^1(x) \oplus SHL^2(x) \oplus ROTL^1(x) \oplus ROTL^6(x)$
$s_2(x) = SHR^2(x) \oplus SHL^1(x) \oplus ROTL^3(x) \oplus ROTL^0(x)$	$s_2(x) = SHR^2(x) \oplus SHL^5(x) \oplus ROTL^{19}(x) \oplus ROTL^7(x)$
$s_3(x) = SHR^2(x) \oplus SHL^2(x) \oplus ROTL^3(x) \oplus ROTL^0(x)$	$s_3(x) = SHR^2(x) \oplus SHL^1(x) \oplus ROTL^{28}(x) \oplus ROTL^4(x)$
$s_4(x) = SHR^1(x) \oplus x$	$s_4(x) = SHR^1(x) \oplus x$
$s_5(x) = SHR^2(x) \oplus x$	$s_5(x) = SHR^2(x) \oplus x$
$r_1(x) = ROTL^1(x)$	$r_1(x) = ROTL^1(x)$
$r_2(x) = ROTL^2(x)$	$r_2(x) = ROTL^2(x)$
$r_3(x) = ROTL^3(x)$	$r_3(x) = ROTL^3(x)$
$r_4(x) = ROTL^0(x)$	$r_4(x) = ROTL^4(x)$
$r_5(x) = ROTL^1(x)$	$r_5(x) = ROTL^5(x)$
$r_6(x) = ROTL^2(x)$	$r_6(x) = ROTL^6(x)$
$r_7(x) = ROTL^3(x)$	$r_7(x) = ROTL^7(x)$

**Table 3.2:** Logic functions used in scaled-down BLUE MIDNIGHT WISH

Having such a small hash outputs, it was easy to analyze and to find collisions for the compression functions of BMW28, BMW32 and BMW48 (but not so easy for BMW64 on our PC with 4GB RAM memory). The average number of calls to the compression functions before finding a collision in a hash of  $n$  bits is given in the Table 3.3. Note that in the second column we give the average number

$\mathcal{A}_n$  of calls to the compression function before finding a collision, and in the third column we give the theoretically expected number  $\mathcal{T}_n$  of calls to the compression function for finding a collision.

$n$	$\mathcal{A}_n$	$\mathcal{T}_n$
28	20,108	20,480
32	84,511	81,920
48	21,469,868	20,971,520
64	/	5,368,709,120

**Table 3.3:** Finding collisions on scaled-down BLUE MIDNIGHT WISH

Beside the attempts of finding collisions we have checked how good is the randomness produced by the compression functions of these heavily scaled-down hash functions. For doing that, for all four variants: BMW28, BMW32, BMW48 and BMW64, we have produced a 500 MBbytes file and examined its randomness by the "TestU01" - a C library for empirical testing of random number generators [17]. The methodology of producing those 500 MBbytes files was the following: We have represented the input message  $M$  as a 64-bits (resp. 128 bits) counter with a starting value 1 and which was increasing by 1. Then the counter  $M$  was represented as 16, 4-bit (resp. 8-bit) variables and we computed  $h = \text{Take}_n\text{LS\_bits}(f_2(M, f_1(M, H)))$ . The values  $h$  were concatenated in order to build a 500 MBbytes file.

Report of TestU01 (applying two test batteries - Rabbit and the NIST FIPS-140-2) for BMW28 is given in Table 3.4 and for BMW32 in Table 3.5. From the reports it is clear that there are certain statistical tests that can distinguish the output of the compression function of BMW28 and BMW32 from an ideal source of randomness. Although the collision analysis for BMW28 and BMW32 are very close to those that are theoretically expected, intuitively it is expectable that such heavily scaled-down instances of the original BLUE MIDNIGHT WISH will be distinguishable from an ideal source of uniformly distributed random bits.

However if we consider that scaling down from 64-bit words to 8-bit words is also very big scaling down, we were surprised to see that BMW48 and BMW64 actually pass all statistical tests from Rabbit and FIPS-140-2 batteries. This actually demonstrates the robustness of BLUE MIDNIGHT WISH design. TestU01 reports (applying again the test batteries - Rabbit and the NIST FIPS-140-2) are given in Table 3.6 and in Table 3.7. BMW48 and BMW64 pass all statistical tests.

<pre> ===== Summary results of Rabbit =====  Version:          TestU01 1.2.1 File:             BMW4Bits28Hash500MB.bin Number of bits:   2139095040 Number of statistics: 40 Total CPU time:   00:10:54.17 The following tests gave p-values outside [0.001, 0.9990]: (eps means a value &lt; 1.0e-300): (eps1 means a value &lt; 1.0e-015):        Test                      p-value -----   1  MultinomialBitsOver        2.8e-05   8  Fourier3                   3.3e-28 10  PeriodsInStrings           3.0e-04 12  HammingCorr, L = 32        1.2e-08 13  HammingCorr, L = 64        8.0e-07 14  HammingCorr, L = 128       4.1e-09 17  HammingIndep, L = 64       7.2e-04 20  Run of bits                 4.4e-04 24  RandomWalk1 H              4.8e-05 25  RandomWalk1 M (L = 1024)   5.2e-04 ----- All other tests were passed </pre>	<pre> ===== Summary results of FIPS-140-2 =====  File:             BMW4Bits28Hash500MB.bin Number of bits:   20000        Test          s-value      p-value   FIPS Decision ----- Monobit            9961          0.71      Pass Poker              6.75          0.96      Pass  0 Runs, length 1: 2501 0 Runs, length 2: 1213 0 Runs, length 3: 603 0 Runs, length 4: 344 0 Runs, length 5: 156 0 Runs, length 6+: 160  1 Runs, length 1: 2467 1 Runs, length 2: 1259 1 Runs, length 3: 614 1 Runs, length 4: 332 1 Runs, length 5: 159 1 Runs, length 6+: 146  Longest run of 0: 14          0.46      Pass Longest run of 1: 13          0.50      Pass ----- All values are within the required intervals of FIPS-140-2 </pre>
---	---

**Table 3.4:** Summary of the TestU01 report for BMW28 (running the Rabbit and FIPS-140-2 battery)

<pre> ===== Summary results of Rabbit =====  Version:          TestU01 1.2.1 File:             BMW4Bits32Hash500MB.bin Number of bits:   2139095040 Number of statistics: 40 Total CPU time:   00:11:07.34 The following tests gave p-values outside [0.001, 0.9990]: (eps means a value &lt; 1.0e-300): (eps1 means a value &lt; 1.0e-015):        Test                      p-value -----   8  Fourier3                   3.6e-30 12  HammingCorr, L = 32        1.7e-14 13  HammingCorr, L = 64        eps 14  HammingCorr, L = 128       7.5e-08 24  RandomWalk1 H              6.6e-05 24  RandomWalk1 J              7.5e-04 25  RandomWalk1 H (L = 1024)   5.5e-04 ----- All other tests were passed </pre>	<pre> ===== Summary results of FIPS-140-2 =====  File:             BMW4Bits32Hash500MB.bin Number of bits:   20000        Test          s-value      p-value   FIPS Decision ----- Monobit            10017         0.41      Pass Poker              9.50          0.85      Pass  0 Runs, length 1: 2533 0 Runs, length 2: 1239 0 Runs, length 3: 605 0 Runs, length 4: 328 0 Runs, length 5: 139 0 Runs, length 6+: 161  1 Runs, length 1: 2479 1 Runs, length 2: 1257 1 Runs, length 3: 650 1 Runs, length 4: 315 1 Runs, length 5: 152 1 Runs, length 6+: 152  Longest run of 0: 13          0.50      Pass Longest run of 1: 14          0.46      Pass ----- All values are within the required intervals of FIPS-140-2 </pre>
--	---

**Table 3.5:** Summary of the TestU01 report for BMW28 (running the Rabbit and FIPS-140-2 battery)

<pre> ===== Summary results of Rabbit =====  Version:      TestU01 1.2.1 File:        BMW8Bits48Hash500MB.bin Number of bits: 2139095040 Number of statistics: 40 Total CPU time: 00:11:05.42  All tests were passed                 </pre>	<pre> ===== Summary results of FIPS-140-2 =====  File:          BMW8Bits48Hash500MB.bin Number of bits: 20000  Test          s-value      p-value      FIPS Decision ----- Monobit       10111           0.06         Pass Poker         6.69            0.97         Pass  0 Runs, length 1: 2493           Pass 0 Runs, length 2: 1247           Pass 0 Runs, length 3: 655            Pass 0 Runs, length 4: 309            Pass 0 Runs, length 5: 142            Pass 0 Runs, length 6+: 145           Pass  1 Runs, length 1: 2464           Pass 1 Runs, length 2: 1272           Pass 1 Runs, length 3: 602            Pass 1 Runs, length 4: 329            Pass 1 Runs, length 5: 149            Pass 1 Runs, length 6+: 175           Pass  Longest run of 0: 11             0.91         Pass Longest run of 1: 14             0.46         Pass ----- All values are within the required intervals of FIPS-140-2                 </pre>
---	---

**Table 3.6:** Summary of the TestU01 report for BMW48 (running the Rabbit and FIPS-140-2 battery)

<pre> ===== Summary results of Rabbit =====  Version:      TestU01 1.2.1 File:        BMW8Bits64Hash500MB.bin Number of bits: 2139095040 Number of statistics: 40 Total CPU time: 00:12:32.89  All tests were passed                 </pre>	<pre> ===== Summary results of FIPS-140-2 =====  File:          BMW8Bits64Hash500MB.bin Number of bits: 20000  Test          s-value      p-value      FIPS Decision ----- Monobit       10030           0.34         Pass Poker         13.89           0.53         Pass  0 Runs, length 1: 2541           Pass 0 Runs, length 2: 1250           Pass 0 Runs, length 3: 614            Pass 0 Runs, length 4: 304            Pass 0 Runs, length 5: 147            Pass 0 Runs, length 6+: 161           Pass  1 Runs, length 1: 2463           Pass 1 Runs, length 2: 1296           Pass 1 Runs, length 3: 643            Pass 1 Runs, length 4: 297            Pass 1 Runs, length 5: 176            Pass 1 Runs, length 6+: 142           Pass  Longest run of 0: 15             0.26         Pass Longest run of 1: 11             0.91         Pass ----- All values are within the required intervals of FIPS-140-2                 </pre>
---	---

**Table 3.7:** Summary of the TestU01 report for BMW64 (running the Rabbit and FIPS-140-2 battery)

## 3.8 Statements about security, support for applications, HMACs and randomized hashing

### 3.8.1 Security statement according to the NIST requirement 4.A.

Security, provided by BLUE MIDNIGHT WISH variants (BMW224, BMW256, BMW384, BMW512) in all applications (standards) is expected to be the same or better than appropriate SHA-2 variants (SHA-224, SHA-256, SHA-384, SHA-512).

### 3.8.2 Statements according to the NIST requirement 4.A.iii.

According to the analysis in previous sections we give a statement of the cryptographic strength of BLUE MIDNIGHT WISH against attacks for finding collisions, preimages, second preimages and resistance to length-extension attacks and multicollision attacks which is summarized in Table 3.8. BLUE MIDNIGHT WISH of message digest size  $n$  ( $n = 224, 256, 384, 512$ ) meet the following security requirements:

- Collision resistance of approximately  $\frac{n}{2}$  bits,
- Preimage resistance of approximately  $n$  bits,
- Second-preimage resistance of approximately  $n - k$  bits for any message shorter than  $2^k$  bits,
- Resistance to length-extension attacks,
- Resistance to multicollision attacks, and
- Any  $m$ -bit hash function specified by taking a fixed subset of the BLUE MIDNIGHT WISH 's output bits meets the above requirements with  $m$  replacing  $n$ .

### 3.8.3 Statement about the support of applications

All BLUE MIDNIGHT WISH variants (BMW224, BMW256, BMW384, BMW512) support wide variety of cryptographic applications, including digital signatures (FIPS 186-2), key derivation (NIST Special Publication 800-56A), hash-based message authentication codes (FIPS 198), deterministic random bit generators (SP 800-90) in the same way as the corresponding SHA-2 variants (SHA-224, SHA-256, SHA-384, SHA-512).

Algorithm abbreviation	Digest size $n$ (in bits)	Work factor for finding collision	Work factor for finding a preimage	Work factor for finding a second preimage of a message shorter than $2^k$ bits	Resistance to length-extension attacks	Resistance to multicollision attacks
BMW224	224	$\approx 2^{112}$	$\approx 2^{224}$	$\approx 2^{224-k}$	Yes	Yes
BMW256	256	$\approx 2^{128}$	$\approx 2^{256}$	$\approx 2^{256-k}$	Yes	Yes
BMW384	384	$\approx 2^{192}$	$\approx 2^{384}$	$\approx 2^{384-k}$	Yes	Yes
BMW512	512	$\approx 2^{256}$	$\approx 2^{512}$	$\approx 2^{512-k}$	Yes	Yes

Table 3.8: Cryptographic strength of the BLUE MIDNIGHT WISH

### 3.8.4 Statement about the special requirements

There are no special requirements when hash function BLUE MIDNIGHT WISH is used to support HMAC, PRF and randomized hashing constructions. All BLUE MIDNIGHT WISH variants (BMW224, BMW256, BMW384, BMW512) are used in these constructions (and in all appropriate standards) in the same way as the corresponding SHA-2 variants (SHA-224, SHA-256, SHA-384, SHA-512).

### 3.8.5 Support of HMAC

BLUE MIDNIGHT WISH is iterative cryptographic hash function. Thus, in combination with a shared secret key can be used in the HMAC standard as it is defined in [18–20].

As the cryptographic strength of HMAC depends on the properties of the underlying hash function, and the conjectured cryptographic strength of BLUE MIDNIGHT WISH is claimed in the Section 3.8.2 here we give a formal statement that BLUE MIDNIGHT WISH can be securely used with the HMAC.

In what follows we are giving 4 examples for every digest size of 224, 256, 384 and 512 bits.

# CHAPTER 3: DESIGN RATIONALES

## BMW224-MAC Test Examples

<pre>Key: 00010203 04050607 08090A0B 0C0D0E0F 10111213 14151617 18191A1B 1C1D1E1F 20212223 24252627 28292A2B 2C2D2E2F 30313233 34353637 38393A3B 3C3D3E3F Key_length: 64 Data: 'Sample #1' Data_length: 9 HMAC: 43817200 C15D1408 B9AC0EFB 77673A2A 7C3CFB6A 95BB00BA 9BD0205C  Key: 30313233 34353637 38393A3B 3C3D3E3F 40414243 Key_length: 20 Data: 'Sample #2' Data_length: 9 HMAC: 60F52403 98DA7CF5 D0661260 7D28F21B 63B90103 0FFA6DFC 9C32E8EA</pre>	<pre>Key: 50515253 54555657 58595A5B 5C5D5E5F 60616263 64656667 68696A6B 6C6D6E6F 70717273 74757677 78797A7B 7C7D7E7F 80818283 84858687 88898A8B 8C8D8E8F 90919293 94959697 98999A9B 9C9D9E9F A0A1A2A3 A4A5A6A7 A8A9AAAB ACADAEAF BOB1B2B3 Key_length: 100 Data: 'The successful verification of a MAC does not completely guarantee that the accompanying message is authentic.' Data_length: 110 HMAC: D72544EF 4FAA6134 E5CCFD73 9A933183 0B8B9E07 C6A7EC97 99ADE48C  Key: 50515253 54555657 58595A5B 5C5D5E5F 60616263 64656667 68696A6B 6C6D6E6F 70717273 74757677 78797A7B 7C7D7E7F 80818283 84858687 88898A8B 8C8D8E8F 90919293 94959697 98999A9B 9C9D9E9F A0A1A2A3 A4A5A6A7 A8A9AAAB ACADAEAF BOB1B2B3 Key_length: 100 Data: 'The successful verification of a MAC does not completely guarantee that the accompanying message is authentic: there is a chance that a source with no knowledge of the key can present a purported MAC.' Data_length: 200 HMAC: 1934A1E8 7F3A527C A2536E98 FEE3D5CA 16340C94 FFAA7733 50BBFB30</pre>
--	---

# CHAPTER 3: DESIGN RATIONALES

## BMW256-MAC Test Examples

<pre>Key: 00010203 04050607 08090A0B 0C0D0E0F 10111213 14151617 18191A1B 1C1D1E1F 20212223 24252627 28292A2B 2C2D2E2F 30313233 34353637 38393A3B 3C3D3E3F Key_length: 64 Data: 'Sample #1' Data_length: 9 HMAC: 86434AF2 C4E1FB07 A5580D8A 4D2EE783 F32A540E 8B95CC3F 11E0B851 7D094DF6  Key: 30313233 34353637 38393A3B 3C3D3E3F 40414243 Key_length: 20 Data: 'Sample #2' Data_length: 9 HMAC: C78B1EE8 EFA86857 A4D9EE05 DE5E3784 7270BFA0 01B748F8 FC3680C6 A5F040CC</pre>	<pre>Key: 50515253 54555657 58595A5B 5C5D5E5F 60616263 64656667 68696A6B 6C6D6E6F 70717273 74757677 78797A7B 7C7D7E7F 80818283 84858687 88898A8B 8C8D8E8F 90919293 94959697 98999A9B 9C9D9E9F A0A1A2A3 A4A5A6A7 A8A9AAAB ACADAEAF BOB1B2B3 Key_length: 100 Data: 'The successful verification of a MAC does not completely guarantee that the accompanying message is authentic.' Data_length: 110 HMAC: DD83D7F9 AA4B23B5 3448FAF1 7AADD6CD 350CF3D3 93A95C36 078E3E19 A193F5EA  Key: 50515253 54555657 58595A5B 5C5D5E5F 60616263 64656667 68696A6B 6C6D6E6F 70717273 74757677 78797A7B 7C7D7E7F 80818283 84858687 88898A8B 8C8D8E8F 90919293 94959697 98999A9B 9C9D9E9F A0A1A2A3 A4A5A6A7 A8A9AAAB ACADAEAF BOB1B2B3 Key_length: 100 Data: 'The successful verification of a MAC does not completely guarantee that the accompanying message is authentic: there is a chance that a source with no knowledge of the key can present a purported MAC.' Data_length: 200 HMAC: C84D3FE8 91BE466A CFAE031E 9F420F0F E34AA375 C0FF2E23 2568D67F 00B3FCE6</pre>
--	---

# CHAPTER 3: DESIGN RATIONALES

## BMW384-MAC Test Examples

<pre> Key: 0001020304050607 08090A0B0C0D0E0F 1011121314151617 18191A1B1C1D1E1F 2021222324252627 28292A2B2C2D2E2F 3031323334353637 38393A3B3C3D3E3F Key_length: 64 Data: 'Sample #1' Data_length: 9 HMAC: B69FB3DF1EFC35F1 2F7764CBFF33719D 16D9326ADDFAEF22 D3070EF9C272D9DC E057F1983F6CD054 6F4B7A5811B27348  Key: 3031323334353637 38393A3B3C3D3E3F 40414243 Key_length: 20 Data: 'Sample #2' Data_length: 9 HMAC: 3A3B108EBAEC7514 C7408260C3F80B19 6F7A7E0E233D48ED C229756A5F243027 FCFF8BED969E09B1 B2F3C6F474439A44         </pre>	<pre> Key: 5051525354555657 58595A5B5C5D5E5F 6061626364656667 68696A6B6C6D6E6F 7071727374757677 78797A7B7C7D7E7F 8081828384858687 88898A8B8C8D8E8F 9091929394959697 98999A9B9C9D9E9F A0A1A2A3A4A5A6A7 A8A9AAABACADAEAF BOB1B2B3B50515253 5455565758595A5B 5C5D5E5F60616263 6465666768696A6B 6C6D6E6F70717273 7475767778797A7B 7C7D7E7F80818283 8485868788898A8B 8C8D8E8F90919293 9495969798999A9B 9C9D9E9FA0A1A2A3 A4A5A6A7A8A9AAAB ACADAEAFBOB1B2B3 Key_length: 200 Data: 'The successful verification of a MAC does not completely guarantee that the accompanying message is authentic.' Data_length: 110 HMAC: A7687B74FBE545D9 F292E74B6E3974FB 7DE1B91C5F7134A6 9F7ABFA2FA3E919E F25EA7F776F935A4 A34FE40543321D26  Key: 5051525354555657 58595A5B5C5D5E5F 6061626364656667 68696A6B6C6D6E6F 7071727374757677 78797A7B7C7D7E7F 8081828384858687 88898A8B8C8D8E8F 9091929394959697 98999A9B9C9D9E9F A0A1A2A3A4A5A6A7 A8A9AAABACADAEAF BOB1B2B3 Key_length: 100 Data: 'The successful verification of a MAC does not completely guarantee that the accompanying message is authentic: there is a chance that a source with no knowledge of the key can present a purported MAC.' Data_length: 200 HMAC: 30C86F32033D447E 98F8EA7E69C0030E 169CE3755DF552BB 26CF32AB3E63ABD5 BFCEE9BCBD9A74CF 25EE03728288B0D9         </pre>
--	--

BMW512-MAC Test Examples

<pre> Key: 0001020304050607 08090A0B0C0D0E0F 1011121314151617 18191A1B1C1D1E1F 2021222324252627 28292A2B2C2D2E2F 3031323334353637 38393A3B3C3D3E3F Key_length: 64 Data: 'Sample #1' Data_length: 9 HMAC: 6C6FA2FF1CB9798F 2560D6E02F1A481E D1F55091BDA0F777 1F5E0CD9B4452A2 564C1B8B04C7E0B6 857450809338A879 BBCFC5563EE29794 3BDE0571497C9351  Key: 3031323334353637 38393A3B3C3D3E3F 40414243 Key_length: 20 Data: 'Sample #2' Data_length: 9 HMAC: AAB97D9E4EA4FED0 CEF6785033FB7EBA 456EE605BA4C4352 EB7150B8F474A216 5FF464365A751CCF 217D09730B2DB6D4 9796416E7965E19A ECDC4A2AED6DAFBE         </pre>	<pre> Key: 5051525354555657 58595A5B5C5D5E5F 6061626364656667 68696A6B6C6D6E6F 7071727374757677 78797A7B7C7D7E7F 8081828384858687 88898A8B8C8D8E8F 9091929394959697 98999A9B9C9D9E9F A0A1A2A3A4A5A6A7 A8A9AAABACADAEAF BOB1B2B350515253 5455565758595A5B 5C5D5E5F60616263 6465666768696A6B 6C6D6E6F70717273 7475767778797A7B 7C7D7E7F80818283 8485868788898A8B 8C8D8E8F90919293 9495969798999A9B 9C9D9E9FA0A1A2A3 A4A5A6A7A8A9AAAB ACADAEAFBOB1B2B3 Key_length: 200 Data: 'The successful verification of a MAC does not completely guarantee that the accompanying message is authentic.' Data_length: 110 HMAC: 9B5592D7A54171D0 49432C6CD640538F 70871534783E0E09 31A1FD06D9717072 E47C50EAB12D6A68 68AC94C8DDB1B7CE 6FEFD58C016105A3 3E880F6498B0B3F5  Key: 5051525354555657 58595A5B5C5D5E5F 6061626364656667 68696A6B6C6D6E6F 7071727374757677 78797A7B7C7D7E7F 8081828384858687 88898A8B8C8D8E8F 9091929394959697 98999A9B9C9D9E9F A0A1A2A3A4A5A6A7 A8A9AAABACADAEAF BOB1B2B3 Key_length: 100 Data: 'The successful verification of a MAC does not completely guarantee that the accompanying message is authentic: there is a chance that a source with no knowledge of the key can present a purported MAC.' Data_length: 200 HMAC: 6B904288E01C5C00 38DFFABA9AAA4C73 039E332BBBE6694B 584058158854B41C 4C82839C7C36646E 91EB81EB1B3B88A9 14184640E6FC1874 E858F9D6F37E6878         </pre>
---	---

### 3.8.6 BLUE MIDNIGHT WISH support of randomized hashing

BLUE MIDNIGHT WISH can be used in the randomizing scheme proposed in [21, 22].

### 3.8.7 Resistance to SHA-2 attacks

BLUE MIDNIGHT WISH is designed to have a security strength that is at least as good as the hash algorithms currently specified in FIPS 180-2, and this security strength is achieved with significantly improved efficiency. Also, BLUE MIDNIGHT WISH is designed so that a possibly successful attack on the SHA-2 hash functions is unlikely to be applicable to BLUE MIDNIGHT WISH .

Is it possible to use any idea from the attacks on SHA-2 (or any other hash function) also to BLUE MIDNIGHT WISH ? Most ideas hardly use the concrete structure and operations of SHA-2. These concrete combinations of sums of variables, concrete operations, shifts, additions, xors, etc. are very important in any concrete attack. And any change, sometimes the tiny change in the design (the shift, xor instead of add, adding another variable) will lead very probably to massively

rebuilding the attack. The change from SHA-2 to BLUE MIDNIGHT WISH is huge. There are different operations and its combinations. All local collisions, neutral bits and so on, hardly created (it was difficult to find even very small result) in known attacks on SHA-2 (SHA-1) are thus ineffective and non-applicable, when used in BLUE MIDNIGHT WISH . No general method is known from the attacks on SHA-2, which would be applicable to BLUE MIDNIGHT WISH .

Most important changes, which have very strong effect in BMW vs. SHA-2:

- a. By using bijections - it guarantees that the change on the input will give the change on the output. There are a lot of bijections in BLUE MIDNIGHT WISH and we found that it is difficult to cancel their influence.
- b. The core of these bijections are non-linear transformations.
- c. By using bijections with good propagation characteristics - all linear and arithmetical bijections, used in BLUE MIDNIGHT WISH are designed to have precise (and good) propagation properties.
- d. 16 summands (operands) in most operations. Unlike many other hash functions where in the compression functions they use basic mixing operation on 4, 5 or 8 operands, BLUE MIDNIGHT WISH in its core uses 16 operands (see the definition of the function  $f_1$ ). It is very difficult to control too much differences in operands of these consecutive operations. Together with the bijective property of these transformations, we have a property that a single differential propagates very fast in the consecutive (iterative) core operations. From this, it follows that to break BLUE MIDNIGHT WISH it is necessary to develop new local collisions, new "rectangular relations", new neutral bits and even new strategies, rather than the old ones used in the analysis and the attacks on SHA-2 or on any other hash function family.



# Estimated Computational Efficiency and Memory Requirements

## 4.1 Speed of BLUE MIDNIGHT WISH on NIST SHA-3 Reference Platform

We have developed and measured the performances of BLUE MIDNIGHT WISH on a platform with the following characteristics:

**CPU:** Intel Core 2 Duo,

**Clock speed:** 2.4 GHz,

**Memory:** 4GB RAM,

**Operating system:** Windows Vista Enterprise 64-bit (x64) Edition with Service Pack 1,

**Compiler:** ANSI C compiler in the Microsoft Visual Studio 2005 Professional Edition.

For measuring the speed of the hash function expressed as cycles/byte we have used the `rdtsc()` function and a modified version of a source code that was given to us by Dr. Brian Gladman from his optimized realization of SHA-2 hash function [23].

### 4.1.1 Speed of the Optimized 32-bit version of BLUE MIDNIGHT WISH

In the Table 4.1 we are giving the speed of all four instances of BLUE MIDNIGHT WISH for the optimized 32-bit version.

	Speed in cycles/byte for different lengths (in bytes) of the digested message.					
MD Size	1	10	100	1000	10,000	100,000
224	2305.00	230.50	42.01	29.28	8.66	8.63
256	781.00	78.10	14.05	9.01	8.69	8.63
384	1945.00	180.10	18.37	13.06	12.72	13.34
512	1789.00	181.30	18.13	13.14	12.72	13.37

**Table 4.1:** The performance of optimized 32-bit version of BLUE MIDNIGHT WISH in machine cycles per data byte on Intel Core 2 Duo for different hash data lengths

	Speed in cycles/byte for different lengths (in bytes) of the digested message.					
MD Size	1	10	100	1000	10,000	100,000
224	1969.00	201.70	36.01	26.28	25.48	7.85
256	613.00	67.30	11.29	8.10	7.83	7.85
384	649.00	70.90	6.85	4.29	4.09	4.06
512	661.00	72.10	7.33	4.27	4.08	4.06

**Table 4.2:** The performance of optimized 64-bit version of BLUE MIDNIGHT WISH in machine cycles per data byte on Intel Core 2 Duo for different hash data lengths

#### 4.1.2 Speed of the Optimized 64-bit version of BLUE MIDNIGHT WISH

In the Table 4.2 we are giving the speed of all four instances of BLUE MIDNIGHT WISH for the optimized 64-bit version.

## 4.2 Memory requirements of BLUE MIDNIGHT WISH on NIST SHA-3 Reference Platform

When processing the message block  $M^{(i)} = (M_0^{(i)}, M_1^{(i)}, \dots, M_{15}^{(i)})$ , we need only the current value of the double pipe  $H^{(i-1)} = (H_0^{(i-1)}, H_1^{(i-1)}, \dots, H_{15}^{(i-1)})$ , two auxiliary words  $XL$  and  $XH$ , and value of the quadruple pipe  $Q^{(i)} = (Q_0^{(i)}, Q_1^{(i)}, \dots, Q_{31}^{(i)})$ .

The need of memory is thus:

- 16 words of  $M^{(i)}$ ,
- 16 words of  $H^{(i)}$ ,

- 2 words  $XL, XH$ ,
- 32 words of  $Q^{(i)}$ .

which is in total 66 words. That means that **BMW224 and BMW256 use 264 bytes** and **BME384 and BMW512 use 528 bytes**.

### 4.3 Estimates for efficiency and memory requirements on 8-bit processors

We have used 8-bit Atmel processors ATmega16 and ATmega64 to test the implementation and performance of the compression function of the two main representatives of the BLUE MIDNIGHT WISH hash function: BMW256 and BMW512. We have used WinAVR – an open source software development tools for the Atmel AVR series of RISC microprocessors and for simulation we have used the AVR Studio v 4.14. In Table 4.3 we are giving the length of the produced executable code and the speed in number of cycles per byte.

Name	Code size (.text + .data + .bootloader) in bytes	Speed (cycles/byte)	8-bit MCU
BMW224/256	10414	1369	ATmega16
BMW384/512	55810	2793	ATmega64

**Table 4.3:** The size and the speed of code for the compression functions for BMW224/256 and BMW384/512

From the analysis of the produced executable code we can project that by direct assembler programming BLUE MIDNIGHT WISH can be implemented in less than 8 Kbytes (BMW256) and in less than 32 Kbytes (BMW512) but this claim will have to be confirmed in the forthcoming period during the NIST competition.

### 4.4 Estimates for a Compact Hardware Implementation

Our initial (non-optimized) VHDL implementation of BLUE MIDNIGHT WISH was done on Xilinx v3200efg1156-8 FPGA. In Table 4.4 we are giving obtained equivalent gate count and also estimates for the compact hardware implementation of the compression function of BLUE MIDNIGHT WISH . These estimates are based on the minimal memory requirements described in Section 4.2.

Name	Obtained equivalent gate count for Xilinx v3200efg1156-8	Estimated gate count for the needed memory	Estimated gate count for the optimized algorithm logic	Estimated minimal total gate count
BMW224/256	44,983	12,672	$\approx 4,000$	$\approx 16,672$
BMW384/512	84,515	25,344	$\approx 6,000$	$\approx 31,344$

**Table 4.4:** Obtained non-optimized gate count for the Xilinx v3200efg1156-8 FPGA, and estimated number of gate count for realization of the compression functions for BMW224/256 and BMW384/512

## 4.5 Internal Parallelizability of BLUE MIDNIGHT WISH

The design of BLUE MIDNIGHT WISH allows very high level of parallelization in computation of its compression function. This parallelism can be achieved by using specifically designed hardware, but with the advent of multicore CPUs, those parts can be computed in different cores in parallel. From the specification given below, we claim that BLUE MIDNIGHT WISH can be computed after 20 “parallel” steps. Of course those 20 “parallel” have different hardware specification and different implementation specifics, but can serve as a general measure of the parallelizability of BLUE MIDNIGHT WISH . The high level parallel specification of BLUE MIDNIGHT WISH is as follows:

### Computing $f_0$

**Step 1:** Computation of all 16 parts of  $W_0^{(i)}, W_1^{(i)}, \dots, W_{15}^{(i)}$  can be done in parallel.

**Step 2:** Computing the values of all 16 parts of  $Q_a$  can be done in parallel.

### Computing $f_1$

**Step 1:** It has 16 expansion steps and each step depends from the previous one. But every expansion step have internal structure that can be parallelized, and a pipelined setup can compute many parts from the next expansion steps that do not depend on the previous expansion value.

### Computing $f_2$

**Step 1:** This step can be actually computed together with the computation of Step 1 of the function  $f_1$ .

**Step 2 (First half):** Computation of the first 8 words  $H_0^{(i)}, H_1^{(i)}, \dots, H_7^{(i)}$  can be done in parallel.

**Step 2 (Second half):** Computation of the last 8 words  $H_8^{(i)}, H_9^{(i)}, \dots, H_{15}^{(i)}$  can be done in parallel.



# Statements

## 5.1 Statement by the Submitter

I, *Svein Johan Knapskog*, do hereby declare that, to the best of my knowledge, the practice of the algorithm, reference implementation, and optimized implementations that I have submitted, known as BLUE MIDNIGHT WISH may be covered by the following U.S. and/or foreign patents: **NONE**.

I do hereby declare that I am aware of no patent applications that may cover the practice of my submitted algorithm, reference implementation or optimized implementations.

I do hereby understand that my submitted algorithm may not be selected for inclusion in the Secure Hash Standard. I also understand and agree that after the close of the submission period, my submission may not be withdrawn from public consideration for SHA-3. I further understand that I will not receive financial compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications relating to my algorithm. I also understand that the U.S. Government may, during the course of the lifetime of the SHS or during the FIPS public review process, modify the algorithm's specifications (e.g., to protect against a newly discovered vulnerability). Should my submission be selected for SHA-3, I hereby agree not to place any restrictions on the use of the algorithm, intending it to be available on a worldwide, non-exclusive, royalty-free basis.

I do hereby agree to provide the statements required by Sections 5.2 and 5.3, below, for any patent or patent application identified to cover the practice of my algorithm, reference implementation or optimized implementations and the right to use such implementations for the purposes of the SHA-3 evaluation process.

I understand that NIST will announce the selected algorithm(s) and proceed to publish the draft

## CHAPTER 5: STATEMENTS

FIPS for public comment. If my algorithm (or the derived algorithm) is not selected for SHA-3 (including those that are not selected for the second round of public evaluation), I understand that all rights, including use rights of the reference and optimized implementations, revert back to the submitter (and other owner[s], as appropriate). Additionally, should the U.S. Government not select my algorithm for SHA-3 at the time NIST ends the competition, all rights revert to the submitter (and other owners as appropriate).

Signed: Svein Johan Knapskog

Title:Prof.

Dated: 24 October 2008

Place: Trondheim, Norway

**5.2 Statement by Patent (and Patent Application) Owner(s)**

**N/A**

### 5.3 Statement by Reference/Optimized Implementations' Owner(s)

We, *Danilo Gligoroski* and *Vlastimil Klima*, are the owners of the submitted reference implementation and optimized implementations and hereby grant the U.S. Government and any interested party the right to use such implementations for the purposes of the SHA-3 evaluation process, notwithstanding that the implementations may be copyrighted.

Signed: Danilo Gligoroski

Title: Prof.

Dated: 24 October 2008

Place: Trondheim, Norway

Signed: Vlastimil Klima

Title: Mr.

Dated: 24 October 2008

Place: Prague, Czech Republic

# References

- [1] *Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family*, 2007. NIST. <http://csrc.nist.gov/groups/ST/hash/index.html>.
- [2] S. Lucks. Design principles for iterated hash functions. Cryptology ePrint Archive, Report 2004/253, 2004. <http://eprint.iacr.org/>.
- [3] S. Lucks. A failure-friendly design principle for hash functions. In *ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 474–494, 2005.
- [4] C. Malinaud J.-S. Coron, Y. Dodis and P. Puniya. Merkle–Damgård revisited: How to construct a hash function. In *CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–440, 2005.
- [5] A. Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In *CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 430–440, 2004.
- [6] J. Kelsey and B. Schneier. Second preimages on  $n$ -bit hash functions for much less than  $2^n$  work. In *EUROCRYPT*, pages 474–490, 2005.
- [7] R. Govaerts B. Preneel and J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *CRYPTO 93*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378, 1994.
- [8] P. Rogaway J. Black and T. Shrimpton. Black-box analysis of the block-cipher-based hash function constructions from pgv. In *In Advances in Cryptology -CRYPTO'02, LNCS vol. 2442, Springer-Verlag*, pages 320–335, 2002.
- [9] E. Filiol. A new statistical testing for symmetric ciphers and hash functions. In *Proceedings, ICICS 2002, LNCS vol. 2513, Springer-Verlag*, pages 342–353, 2002.

## REFERENCES

- [10] M.-J. O. Saarinen. Chosen-iv statistical attacks on estream ciphers. In *Proceeding of SECRYPT 2006*, pages 260–266, 2006.
- [11] T. Johansson H. Englund and M. S. Turan. A framework for chosen iv statistical analysis of stream ciphers. pages 268–281, 2007.
- [12] S. O’neil. Algebraic structure defectoscopy, 2007. <http://eprint.iacr.org/2007/378>.
- [13] T. A. Berson. Differential cryptanalysis mod  $2^{32}$  with applications to md5. In *Proceedings of EUROCRYPT 92*, pages 71–80, 1992.
- [14] H. Lipmaa and S. Moriai. Efficient algorithms for computing differential properties of addition. In *Proceedings of FSE 2001*, pages 336–350. Springer-Verlag, 2002.
- [15] H. Lipmaa, J. Wallén, and P. Dumas. On the Additive Differential Probability of Exclusive-Or. In *Fast Software Encryption 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 317–331. Springer-Verlag, 2004.
- [16] S. Paul and B. Preneel. Solving systems of differential equations of addition. In *Proceedings of ACISP 2005*, volume 3574 of *Lecture Notes in Computer Science*, pages 75–88, 2005.
- [17] P. L’Ecuyer and R. Simard. Testu01: A c library for empirical testing of random number generators. *ACM Trans. Math. Softw.*, 33(4):22, 2007.
- [18] M. Bellare H. Krawczyk and R. Canetti. Rfc2104 - hmac: Keyed-hashing for message authentication, 1997. <http://www.faqs.org/rfcs/rfc2104.html>.
- [19] American Bankers Association. Keyed hash message authentication code, 2000.
- [20] National Institute of Standards and Technology. The keyed-hash message authentication code (hmac), March 6, 2002. <http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf>.
- [21] S. Halevi and H. Krawczyk. Strengthening digital signatures via randomized hashing, 2006.
- [22] National Institute of Standards and Technology. Randomized hashing for digital signatures, August, 2008. [http://csrc.nist.gov/publications/drafts/800-106/2nd-Draft\\_SP800-106\\_July2008.pdf](http://csrc.nist.gov/publications/drafts/800-106/2nd-Draft_SP800-106_July2008.pdf).
- [23] B. Gladman. Sha1, sha2, hmac and key derivation in c. [http://fp.gladman.plus.com/cryptography\\_technology/sha/index.htm](http://fp.gladman.plus.com/cryptography_technology/sha/index.htm).