



Blue Midnight Wish

Skript zum Seminarthema
„Blue Midnight Wish — Ein SHA-3 Kandidat“

Sommersemester 2010
an der Humboldt Universität zu Berlin.

Daniel Schliebner

Herausgabe: 11. November 2010

Inhaltsverzeichnis

1. Einführung	2
1.1 Einleitung	2
1.2 Überblick	2
1.3 Vorbetrachtungen	3
2. Algorithmus	5
2.1 Blue Midnight Wish	5
2.2 Preprocessing	6
2.3 Hashberechnung	6
2.4 Finalisation	11
3. Kryptoanalyse	11
3.1 Sicherheitsrelevante Komponenten	11
3.2 BMW als PGV-Schema	12
3.3 Sicherheitsannahmen	13
3.4 Weitere Eigenschaften von BMW	14
A. Anhang	16
Literaturverzeichnis	16

1. Einführung

1.1 Einleitung

Diese Arbeit umfasst eine Ausarbeitung des von mir im Sommersemester 2010 gehaltenen Vortrages zum Thema „Blue Midnight Wish - Ein SHA-3 Kandidat“.

Das **NIST** (National Institute of Standards and Technology) eröffnete dazu am 1. November 2008 einen Wettbewerb (Cryptographic Hash Algorithm Competition), welcher es zum Ziel hat, einen kryptographischen Hashalgorithmus zu finden, der letztendlich als SHA-3 Algorithmus und Nachfolger von SHA-2 standardisiert wird:

„NIST has opened a public competition to develop a new cryptographic hash algorithm, which converts a variable length message into a short “message digest” that can be used for digital signatures, message authentication and other applications. The competition is NIST’s response to recent advances in the cryptanalysis of hash functions. The new hash algorithm will be called “SHA-3” and will augment the hash algorithms currently specified in FIPS 180-2, Secure Hash Standard.“ [1].

Die erste Runde dieses Wettbewerbes (an dem zahlreiche Universitäten und Wissenschaftler beteiligt sind und waren) begann am 1. November 2008 mit insgesamt 64 Kandidaten, welche sich schnell (bis zum 09. Dezember 2008) auf 51 ernst zu nehmende Kandidaten reduzierten. Am 24. Juli 2009 wurde schließlich Runde 2 des Wettbewerbes mit lediglich 14 verbliebenen Kandidaten gestartet, darunter auch **Blue Midnight Wish (BMW)**, ein auf der Merkle-Damgård-Konstruktion basierender Hash-Algorithmus. Thema dieser Arbeit ist die Beschreibung und Erläuterung dieses Hash-Algorithmus, basierend auf den Original-Dokumenten [2] der Autoren Danilo Gligoroski, Vlastimil Klima und Svein Johan Knapskog.

1.2 Überblick

Der in dieser Arbeit diskutierte BMW-Algorithmus wurde hauptsächlich von Danilo Gligoroski, Vlastimil Klima und Svein Johan Knapskog aber nicht zuletzt auch von Mohamed El-Hadedy, Jørn Amundsen und Stig Frode Mjølsnes an der NTNU (Norwegian University of Science and Technology) entwickelt. BMW ist ein auf der Merkle-Damgård-Konstruktion basierender Hash-Algorithmus, der darüber hinaus eine erweiterte interne Pipe der Größe $\omega \geq n$ benutzt, wobei wir hier mit $n \in \mathbb{N}$ die Digestgröße bezeichnen möchten (genauere Informationen in Abschnitt 1.3) - daher also einen sogenannten **Wide-Pipe-Hash** darstellt.

Der BMW-Algorithmus wird in vier möglichen Varianten beschrieben:

BMW224, BMW256, BMW384 und BMW512.

Hierbei bezeichnet die Nummer jeweils die Größe des Hash-Digest in Bit (siehe Abschnitt 2 für detaillierte Informationen zu den einzelnen Varianten). Nicht zuletzt unerwähnt bleiben soll auch die Namensfindung des Algorithmus. Das Problem des ersten Arbeitstitels (Blue Wish) war, dass dieser bereits eine registrierte Marke einer Firma für Handtücher war,

wodurch ein anderer Name gefunden werden musste; nun kam es dazu, dass eine um die Mitternachtszeit herum produzierte Version des Algorithmus die besten Charakteristiken aufwies. Dies gab den Ausschlag für die Wahl des Namens Blue Midnight Wish.

1.3 Vorbetrachtungen

Wir wollen zu Beginn eine Reihe von Bezeichnungen fixieren.

- Definition 1**
- \mathcal{M} sei die zu hashende Nachricht der Länge ℓ Bits;
 - n Bits sei der Hash-Digest (d. h. die Länge der Hash-Nachricht);
 - m sei die Block-Größe innerhalb der Merkle-Damgård-Konstruktion;
 - N sei die Anzahl der Blöcke innerhalb der gepaddeten Nachricht;
 - $\mathcal{M}^{(i)}$ sei der i -te m -Bit Block von \mathcal{M} ;
 - $\mathcal{M}_j^{(i)}$ sei das j -te Wort des i -ten Nachrichtenblocks.

Für Operationen auf Bit und/oder Byte-Ebene fixieren wir die folgenden Bezeichnungen.

Definition 2 Wir schreiben

- $\text{SHL}^r(x)$ für einen Links-Shift um r Bit,
- $\text{SHR}^r(x)$ für einen Rechts-Shift um r Bit,
- $\text{ROTL}^r(x)$ für einen zirkulären Links-Shift,
- $+$ und $-$ für eine Addition/Substraktion modulo $2^{32}/2^{64}$,
- \oplus für eine bitweise XOR-Operation auf Wörtern,

wobei x im allgemeinen ein 32- oder 64-Bit Wort ist.

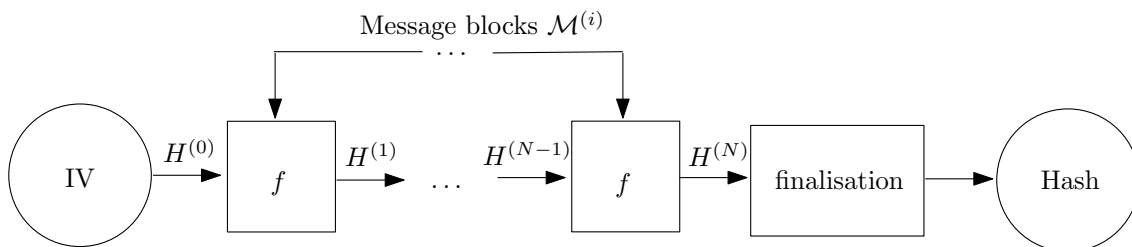


Abbildung 1.1: Merkle Damgård Konstruktion.

Um detailliert auf das Wide-Pipe-Design und seine Idee eingehen zu können, erinnern wir zunächst an die Merkle-Damgård-Konstruktion, welche basierend auf einer Kompressionsfunktion $f : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}^n$ iterativ einen Hash-Wert zu einer durch ein Padding auf die Länge $N \cdot m$ gebrachte Nachricht \mathcal{M} auf die Länge n hasht. Grafik 1.1 veranschaulicht das iterative Vorgehen. Ausgehend von einem Initialisierungsvektor $IV \in \{0, 1\}^n$ wird iterativ der Hashwert vermöge der Vorschrift

$$H^{(i+1)} := f(H^{(i)}, M^{(i)}), \quad i \geq 0$$

berechnet und als Digest $H^{(N)}$ ausgegeben.

Jedoch stellte sich diese klassische Variante als problematisch dar. So fand Joux [3] eine Multikollisions-Angriffe (eine sogenannte K -Kollision, d. h. K Nachrichten mit dem gleichen Hashwert) gegen den pragmatischen iterativen Hash der Merkle-Damgård-Konstruktion, welche mit Zeitaufwand $\Omega(\log(K) \cdot 2^{n/2})$ statt mit $\Omega(2^{(K-1)n/K})$ einer idealen Hashfunktion durchzuführen war. Die Idee der Angriffe bestand darin, im Falle $K = 2^N$ insgesamt N lokale Kollisionen $M_0^{(i)} = M_1^{(i)}$ der Form $f(H^{(i-1)}, M_0^{(i)}) = f(H^{(i-1)}, M_1^{(i)})$ zu berechnen. Diese Angriffe war der Anlass, dass Lucks [4] 2004 vorschlug, die interne Pipe der Merkle-Damgård-Konstruktion auf $\omega \geq 2n$ Bit auszuweiten.

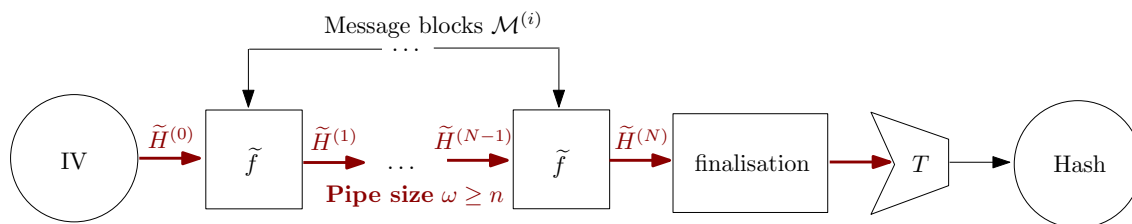


Abbildung 1.2: Erweiterte Pipe in der Merkle-Damgård-Konstruktion.

Durch diese Ausweitung der internen Pipe führte er eine modifizierte Kompressionsfunktion \tilde{f} zusammen mit einer zusätzlichen Kompressionsfunktion $T : \{0, 1\}^\omega \rightarrow \{0, 1\}^n$ in die Merkle-Damgård-Konstruktion ein (vgl. Grafik 1.2). Der zwischenzeitliche Hashwert $\tilde{H}^{(N)} \in \{0, 1\}^\omega$ wird dabei auch als **Intermediate Hash** bezeichnet und die soeben beschriebene Modifikation **Iterativer Wide-Pipe Hash**. Darüber hinaus bewies Lucks die folgenden beiden Theoreme [4]

Theorem 1 *Um sicherzustellen, dass der Iterative Wide-Pipe Hash mit interner Pipe der Größe ω asymptotisch so sicher gegen Multikollisionsangriffe ist, wie ein idealer Hash, ist $\omega \geq 2n$ notwendig.*

Darüber hinaus bewies er

Theorem 2 *Iterative Wide-Pipe Hashes mit interner Pipe der Größe $\omega \geq 2n$ sind sicher gegen alle generischen Angriffe (d. h. sie sind asymptotisch genauso sicher wie ein idealer Hash).*

2. Algorithmus

2.1 Blue Midnight Wish

Wie bereits in Abschnitt 1 erwähnt, existieren von BMW vier Varianten mit unterschiedlichem Hash-Digest n . Die nachfolgende Tabelle gibt einen kurzen Überblick über alle vier Varianten.

Algorithmus	Nachrichtenlänge ℓ	Blockgröße m	Wortgröße w	Digest n
BMW224	$< 2^{64}$	512	32	224
BMW256	$< 2^{64}$	512	32	256
BMW384	$< 2^{64}$	1024	64	384
BMW512	$< 2^{64}$	1024	64	512

Die Algorithmen BMW224 und BMW256 sind entsprechend für den Einsatz auf 32-Bit Architekturen, BMW384 und BMW512 dagegen für 64-Bit Architekturen geeignet. Ein wichtiges Detail ist darüber hinaus die verwendete Endianess. Analog zu anderen Kandidaten der Merkle-Damgård-Klasse verwendet auch BMW auf Bitebene Big-Endian und auf Byte-Ebene Low-Endian.

Um einen ersten Überblick über den Algorithmus zu bekommen, geben wir nachfolgend seine komplette Spezifikation an, wobei wir auf die einzelnen vorkommenden Funktionen und Konstanten in den folgenden Abschnitten eingehen werden.

Algorithmus 1 (Blue Midnight Wish) 0. *Input:* \mathcal{M} with $|\mathcal{M}| = \ell$, hash size n .

1. Preprocessing

- (a) Padde Nachricht \mathcal{M} .
- (b) Parse gepadde Nachricht in N m -Bit Blöcke $M^{(1)}, \dots, M^{(N)}$.
- (c) Setze Initialisierungsvektor $H^{(0)}$ (m -Bit).

2. Hashberechnung

```

for  $i = 1$  to  $N$ 
   $Q_a^{(i)} = f_0(M^{(i)}, H^{(i-1)});$ 
   $Q_b^{(i)} = f_1(M^{(i)}, H^{(i-1)}, Q_a^{(i)});$ 
   $H^{(i)} = f_2(M^{(i)}, Q_a^{(i)}, Q_b^{(i)});$ 
endfor

```

3. Finalisation

```

 $Q_a^{\text{final}} = f_0(H^N, \text{F\_CONST});$ 
 $Q_b^{\text{final}} = f_1(H^N, \text{F\_CONST}, Q_a^{\text{final}});$ 
 $H^{\text{final}} = f_2(H^N, Q_a^{\text{final}}, Q_b^{\text{final}});$ 

```

4. *Output:* n -LSB(H^{final}) ($= H_8^N || \dots || H_{15}^N$).

2.2 Preprocessing

Das Preprocessing ist im BMW-Algorithmus pragmatisch und für die Merkle-Damgård-Konstruktion typisch umgesetzt. Bezugnehmend auf den im vorherigen Abschnitt definierten Algorithmus Blue Midnight Wish betrachten wir zunächst Schritt (a) im Preprocessing. Hier wird im Padding der Nachricht \mathcal{M} ein neuer String erzeugt, auf Basis dessen schließlich der Hash berechnet wird. Das Resultat des Paddings hat dabei schlicht die Form

$$\mathcal{M}||1||\underbrace{00 \dots 00}_{k \text{ times}}||\text{bin}(\ell),$$

wobei $\ell + k + 1 \equiv_{512} 448$ (oder $\ell + k + 1 \equiv_{1024} 960$) erfüllt sein muss. Als Effekt erhalten wir somit einen String einer Länge $N \cdot m$. Dies ist notwendig um im Folgenden auf Textblöcken der Größe m arbeiten zu können. Diese werden im Schritt (b) des Preprocessing erzeugt, indem der i te Block $M^{(i)}$ aus den Bits $im, \dots, i(m+1) - 1$, $i = 1, \dots, N$ der gepaddeten Nachricht aus (a) besteht.

Für Schritt (c) muss ein Initialisierungsvektor $H^{(0)} \in \{0, 1\}^m$ festgelegt werden, welcher offensichtlich von der Blockgröße abhängt, welche wiederum durch die verwendete Variante des BMW-Algorithmus vorgegeben wird. Je nachdem werden die folgenden Initialisierungsvektoren benutzt:

- $n = 224$: zerlege $0x00 \dots 0x3F$ in sechzehn 32-Bit Worte.
- $n = 256$: zerlege $0x40 \dots 0x7F$ in sechzehn 32-Bit Worte.
- $n = 384$: zerlege $0x00 \dots 0x7F$ in sechzehn 64-Bit Worte.
- $n = 512$: zerlege $0x80 \dots 0xFF$ in sechzehn 64-Bit Worte.

2.3 Hashberechnung

Blue Midnight Wish ist ein Algorithmus basierend auf der Merkle-Damgård Konstruktion. Entsprechend basiert die Berechnung des Hash-Digest auf der iterierten Anwendung einer Kompressionsfunktion

$$f : \{0, 1\}^k \longrightarrow \{0, 1\}^m,$$

wobei das $k \in \mathbb{N}$ ein Vielfaches der Breite der erweiterten internen Pipe gemäß des Wide-Pipe Designs von Lucks beschreibt. Konkret wird im BMW-Algorithmus, verweisend auf die Sicherheitstheoreme (Theorem 1 und 2), [1] der Wert

$$\omega = m = 2n$$

gewählt (siehe Tabelle der BMW Varianten in Abschnitt 2.1), was beweisbar theoretisch optimale Sicherheitseigenschaften nach sich zieht (in Hinblick auf generische Attacken).

Dementsprechend verwendet der BMW-Algorithmus, schematisch dargestellt in Grafik 2.3, eine Kompressionsfunktion $f : \{0, 1\}^{2m} \longrightarrow \{0, 1\}^m$, welche als Superposition $f = f_2 \circ f_1 \circ f_0$ von drei Funktionen f_1, f_2, f_3 wie folgt beschrieben wird:

$$\begin{aligned} f_0 &: \{0, 1\}^{2m} \longrightarrow \{0, 1\}^m, \\ f_1 &: \{0, 1\}^{3m} \longrightarrow \{0, 1\}^m, \\ f_2 &: \{0, 1\}^{3m} \longrightarrow \{0, 1\}^m. \end{aligned}$$

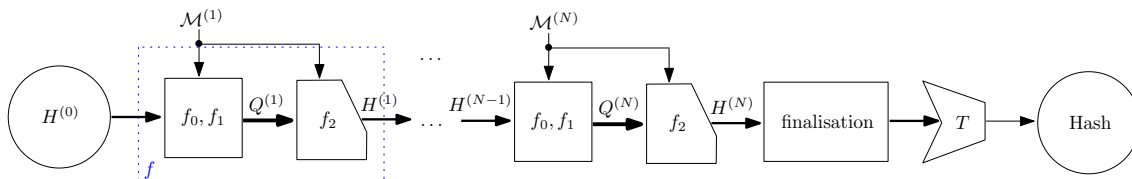


Fig.: The BMW algorithm

Abbildung 2.3: Schematische Darstellung des BMW-Algorithmus.

Man beachte hierbei, dass die gewählte Schreibweise $f = f_2 \circ f_1 \circ f_0$ mathematisch nicht ganz korrekt ist, jedoch die Intuition beschreibt: von f_0 berechnete Bit-Strings werden (eingeteilt in m -Bit Blöcke) innerhalb von f_1 verwendet, welche wiederum Bitblöcke bereit stellt, die schließlich von f_2 benutzt werden.

Wir werden sehen, dass die Sicherheit des Algorithmus hauptsächlich auf nicht-linearen Ausdrücken und Permutationen innerhalb der Funktionen f_0, f_1 und f_2 beruht. Dazu werden wir diese Funktionen nun eingehend beschreiben und verwenden dabei die Notationen der Entwickler [1], aus deren Dokumentation auch die folgende Tabelle in Grafik 2.4, welche einen Überblick über zusätzlich verwendete Funktionen und Bezeichnungen gibt, entnommen wurde.

Definition von f_0

Die Funktion

$$f_0 : \{0, 1\}^{2m} \longrightarrow \{0, 1\}^m$$

berechnet auf Grundlage der m -Bit Blöcke $M^{(i)}$ und $H^{(i-1)}$ den folgenden m -Bit-Block. Wir definieren $f_0(M^{(i)}, H^{(i-1)}) := Q_a^{(i)}$, wobei

$$Q_a^{(i)} := A_2(A_1(M^{(i)} \oplus H^{(i-1)})) + \text{ROTL}^1(H^{(i-1)}).$$

Hierbei ist $A_1 \in \{0, 1, -1\}^{16 \times 16}$ eine Matrix, welche aus einer Matrix $A'_1 \in \mathbb{F}_2^{16 \times 16}$ entsteht, indem zufällig Werte '1' in '-1' so geändert werden, dass $\det A_1 \in \mathbb{Z}_{2^w}^\times$. A'_1 hat dabei die

folgende Gestalt

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

Die Transformation A_2 wird dann definiert durch

$$A_2(W_j^{(i)}) := s_{j \bmod 5}(W_j^{(i)}) + H_{(j+1) \bmod 16}^{(i-1)},$$

wobei $W^{(i)} := A_1(M^{(i)} \oplus H^{(i-1)})$. Die Transformationen s_0, s_1, s_2, s_3, s_4 sind in Grafik 2.4 dargestellt und basieren hauptsächlich auf den Transformationen ROTL, SHL und SHR. Im Anhang findet man eine konkrete Darstellung der Funktion f_0 .

Definition von f_1

Die Funktion

$$f_1 : \{0, 1\}^{3m} \longrightarrow \{0, 1\}^m$$

verwendet zur Berechnung des zweiten m -Bit Blocks $Q_b^{(i)}$ den vermöge der Funktion f_0 berechneten Block $Q_a^{(i)}$. Konkret definiert sich f_1 durch die folgende Vorschrift

$$f_1(M^{(i)}, H^{(i-1)}, Q_a^{(i)}) := Q_b^{(i)}$$

mit $Q_b^{(i)} = (Q_{16}^{(i)}, \dots, Q_{31}^{(i)})$, wobei sich die $Q_j^{(i)}$, $j = 16, \dots, 31$ mithilfe der Wörter $Q_a^{(i)}$ wiefolgt errechnen (man beachte, dass die Wörter aus $Q_a^{(i)}$ implizit innerhalb der in Grafik 2.4 definierten Funktionen expand_1 und expand_2 vorkommen, f_1 also tatasächlich von $Q_a^{(i)}$ abhängt).

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

Abbildung 2.4: Logische Funktionen in f_0 , f_1 , und f_2 . (Tabelle aus [2, S. 8].)

Wir berechnen die Wörter $Q_j^{(i)}$, $j = 16, \dots, 31$ nun wie folgt.

for $ii = 0$ **to** $ExpandRounds_1 - 1$

$$Q_{ii+16}^{(i)} = expand_1(ii + 16);$$

for $ii = ExpandRounds_1$ **to** $ExpandRounds_1 + ExpandRounds_2 - 1$

$$Q_{ii+16}^{(i)} = expand_2(ii + 16);$$

Hierbei bezeichnen $ExpandRounds_1$ und $ExpandRounds_2$ Konstanten, welche gemäß der Dokumentation zu BMW mit den Werten $ExpandRounds_1 = 2$ und $ExpandRounds_2 = 14$ gewählt werden sollten (die Wahl dieser Werte ergibt dann den bestmöglichen Kompromiss im Trade-Off zwischen Geschwindigkeit und Sicherheit, siehe [1]). Als Resultat erhalten eine Quadrupel-Pipe durch das soeben berechnete Tupel $(Q_a^{(i)}, Q_b^{(i)})$. Dieses wird nun mithilfe der Funktion f_2 auf einen m -Bit String zusammen gestaucht („gefaltet“).

Definition von f_2

Die Funktion

$$f_2 : \{0, 1\}^{3m} \longrightarrow \{0, 1\}^m$$

berechnet den finalen Intermediate Hash durch Folding, d. h. die durch die Funktionen f_0 und f_1 ermittelte Double-Pipe, repräsentiert durch das Quadrupel $(Q_a^{(i)}, Q_b^{(i)})$, wird auf die Größe m zusammen gestaucht:

$$f_2(M^{(i)}, Q_a^{(i)}, Q_b^{(i)}) := H^{(i)}.$$

Dies erfolgt im Wesentlichen mit einer großen Anzahl von XOR-, Shift-, Rotations- und Additions-Operationen (wobei die Addition wie bereits zuvor modulo 2^w durchgeführt wird). Eine konkrete Darstellung zeigt die folgende Grafik.

1. Compute the 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 $H^{(i)}$:

$H_0^{(i)} =$	$(SHL^5(XH) \oplus SHR^5(Q_{16}^{(i)}) \oplus M_0^{(i)}) +$	$(XL \oplus Q_{24}^{(i)} \oplus Q_0^{(i)})$
$H_1^{(i)} =$	$(SHR^7(XH) \oplus SHL^8(Q_{17}^{(i)}) \oplus M_1^{(i)}) +$	$(XL \oplus Q_{25}^{(i)} \oplus Q_1^{(i)})$
$H_2^{(i)} =$	$(SHR^5(XH) \oplus SHL^5(Q_{18}^{(i)}) \oplus M_2^{(i)}) +$	$(XL \oplus Q_{26}^{(i)} \oplus Q_2^{(i)})$
$H_3^{(i)} =$	$(SHR^1(XH) \oplus SHL^5(Q_{19}^{(i)}) \oplus M_3^{(i)}) +$	$(XL \oplus Q_{27}^{(i)} \oplus Q_3^{(i)})$
$H_4^{(i)} =$	$(SHR^3(XH) \oplus Q_{20}^{(i)} \oplus M_4^{(i)}) +$	$(XL \oplus Q_{28}^{(i)} \oplus Q_4^{(i)})$
$H_5^{(i)} =$	$(SHL^6(XH) \oplus SHR^6(Q_{21}^{(i)}) \oplus M_5^{(i)}) +$	$(XL \oplus Q_{29}^{(i)} \oplus Q_5^{(i)})$
$H_6^{(i)} =$	$(SHR^4(XH) \oplus SHL^6(Q_{22}^{(i)}) \oplus M_6^{(i)}) +$	$(XL \oplus Q_{30}^{(i)} \oplus Q_6^{(i)})$
$H_7^{(i)} =$	$(SHR^{11}(XH) \oplus SHL^2(Q_{23}^{(i)}) \oplus M_7^{(i)}) +$	$(XL \oplus Q_{31}^{(i)} \oplus Q_7^{(i)})$
$H_8^{(i)} = ROTL^9(H_4^{(i)}) +$	$(XH \oplus Q_{24}^{(i)} \oplus M_8^{(i)}) +$	$(SHL^8(XL) \oplus Q_{23}^{(i)} \oplus Q_8^{(i)})$
$H_9^{(i)} = ROTL^{10}(H_5^{(i)}) +$	$(XH \oplus Q_{25}^{(i)} \oplus M_9^{(i)}) +$	$(SHR^6(XL) \oplus Q_{16}^{(i)} \oplus Q_9^{(i)})$
$H_{10}^{(i)} = ROTL^{11}(H_6^{(i)}) +$	$(XH \oplus Q_{26}^{(i)} \oplus M_{10}^{(i)}) +$	$(SHL^6(XL) \oplus Q_{17}^{(i)} \oplus Q_{10}^{(i)})$
$H_{11}^{(i)} = ROTL^{12}(H_7^{(i)}) +$	$(XH \oplus Q_{27}^{(i)} \oplus M_{11}^{(i)}) +$	$(SHL^4(XL) \oplus Q_{18}^{(i)} \oplus Q_{11}^{(i)})$
$H_{12}^{(i)} = ROTL^{13}(H_0^{(i)}) +$	$(XH \oplus Q_{28}^{(i)} \oplus M_{12}^{(i)}) +$	$(SHR^3(XL) \oplus Q_{19}^{(i)} \oplus Q_{12}^{(i)})$
$H_{13}^{(i)} = ROTL^{14}(H_1^{(i)}) +$	$(XH \oplus Q_{29}^{(i)} \oplus M_{13}^{(i)}) +$	$(SHR^4(XL) \oplus Q_{20}^{(i)} \oplus Q_{13}^{(i)})$
$H_{14}^{(i)} = ROTL^{15}(H_2^{(i)}) +$	$(XH \oplus Q_{30}^{(i)} \oplus M_{14}^{(i)}) +$	$(SHR^7(XL) \oplus Q_{21}^{(i)} \oplus Q_{14}^{(i)})$
$H_{15}^{(i)} = ROTL^{16}(H_3^{(i)}) +$	$(XH \oplus Q_{31}^{(i)} \oplus M_{15}^{(i)}) +$	$(SHR^2(XL) \oplus Q_{22}^{(i)} \oplus Q_{15}^{(i)})$

Abbildung 2.5: Die Funktion f_2 . (Tabelle aus [2, S. 20].)

2.4 Finalisation

Die Finalisation ist typisch für Merkle-Damgård-Konstruktionen. Ihr Ziel ist es vor allem, den letzten iterativ berechneten Intermediate Hash noch einmal so zu modifizieren, dass etwa das Auffinden von Pseudo-Kollisionen (d. h. Kollisionen in einzelnen Bits des Hash-Digest) erschwert wird (wie es auch hier der Fall ist). Nicht zuletzt ist zu beachten, dass in der Finalisation von BMW (siehe Schritt 3 im Algorithmus) die Rollen des Intermediate Hash $H^{(N)}$ und der Quadrupel-Pipe $(Q_a^{(i)}, Q_b^{(i)})$ vertauscht wurden, sowie im zweiten Argument der Funktionen f_0 und f_1 konstante Werte benutzt wurden. Die Intuition hinter diesen Maßnahmen liegt darin, potentiellen Gegnern zu erschweren, (Pseudo-)Urbilder oder (Pseudo-)Kollisionen zu finden, da die Zahl der Freiheitsgrade im ggf. frei wählbaren Hash-Wert des Gegners reduziert wird. Die verwendeten Konstanten im BMW-Algorithmus sind dabei die folgenden.

$$\begin{aligned} n \in \{224, 256\} : \quad & \text{F_CONST} = (\text{0xaaaaaaaa0}, \dots, \text{0xaaaaaaaaaf}), \\ n \in \{384, 512\} : \quad & \text{F_CONST} = (\text{0xaaaaaaaaaaaaaaaa0}, \dots, \text{0xaaaaaaaaaaaaaaaaaf}). \end{aligned}$$

3. Kryptoanalyse

3.1 Sicherheitsrelevante Komponenten

Die Sicherheit des BMW Algorithmus stützt sich im Wesentlichen auf die drei folgenden verwendeten Techniken und Komponenten.

- (A) Das Double-Pipe-Design.
- (B) Eine hohe Anzahl von Permutationen und Bijektionen in den Funktionen f_0 , f_1 und f_2 .
- (C) Eine hohe Anzahl von nicht-linearen Operationen innerhalb der Funktionen f_0 , f_1 und f_2 .

In welcher Hinsicht Punkt (A) relevant für die Sicherheit ist, haben wir bereits in Abschnitt 1.3 (Theorem 1 und Theorem 2) erläutert. Konkret erreichen die Autoren des Blue Midnight Wish dadurch eine beweisbare Sicherheit gegen generische Attacken (d. h. solche Attacken, welche auch anwendbar bleiben, sofern die Kompressionsfunktion f des BMW-Algorithmus gegen ein Zufallsorakel ausgetauscht wird).

Hinsichtlich Punkt (B) haben die Autoren von Blue Midnight Wish das folgende Theorem bewiesen

Theorem 3 1. Ist $H^{(i-1)}$ fixiert, so ist $f_0(\cdot, H^{(i-1)})$ eine Bijektion.

2. Für gegebenes $H^{(i-1)}$ ist die Funktion $f_1(\cdot, H^{(i-1)}, \cdot)$ eine Multipermutation.

3. Sind $Q_b^{(i)}$ und $M^{(i)}$ fixiert, so ist $f_2(M^{(i)}, \cdot, Q_b^{(i)})$ eine Bijektion.

4. Sind $Q_b^{(i)}$ und $Q_a^{(i)}$ fixiert, so ist $f_2(\cdot, Q_a^{(i)}, Q_b^{(i)})$ eine Bijektion.

Beweis: [1, S. 35]

□

Dies zeigt, dass in der Tat eine hohe Anzahl an Permutationen und Bijektionen innerhalb der Kompressionsfunktion vorkommen. Allgemein ist dies ein Indiz für die Beständigkeit der Hashfunktion gegen differentielle Kryptoanalyse.

Punkt (C) spricht die Operationen $+$ und $-$ in $\mathbb{Z}_{2^{32}}$ (oder $\mathbb{Z}_{2^{64}}$), wie sie innerhalb der Funktionen f_0 , f_1 und f_2 verwendet werden, an. Diese sind innerhalb des Körpers $\text{GF}(2^{32})$ (oder $\text{GF}(2^{64})$) nicht linear. Nicht zuletzt auch diese (hohe Anzahl an) Operationen erschweren Gegnern das gezielte Manipulieren des Hashwertes mit dem Ziel, (Pseudo-)Kollisionen und (Pseudo)-Urbilder zu bestimmen, da die lineare Approximation durch XORs nicht effizient möglich ist.

Die soeben angesprochenen Begründungen und sicherheitsrelevanten Aspekte in den Punkten (A) bis (C) werden in Abschnitt 3.3 noch einmal konkretisiert.

3.2 BMW als PGV-Schema

Preneel, Govaerts und Vandewalle haben Hash-Funktionen, welche auf Block-Chiffren basieren, innerhalb von 12 Schemata klassifiziert [5] (PGV1 - PGV12) und darüber hinaus diverse Sicherheitseigenschaften überprüft. Wir werden sehen, dass der BMW-Algorithmus als verallgemeinertes PGV6-Schema aufgefasst werden kann, was dazu führt, dass die Autoren sich auf bereits bekannte Sicherheitseigenschaften der PGV6-Konstruktion berufen können (für Details dazu siehe [5]). Allgemein wird im PGV6-Schema der Intermediate Hash wie folgt ermittelt

$$H^{(i)} = E(H^{(i-1)}, M^{(i)} \oplus H^{(i-1)}) \oplus M^{(i)} \oplus H^{(i-1)}.$$

Dies gibt uns nun die Möglichkeit, das folgende Theorem zu formulieren (man beachte, dass die Autoren BMW zum PGV1 Schema verallgemeinern [1, S. 36f.]).

Theorem 4 *Der BMW Hash kann als verallgemeinertes PGV6 Schema aufgefasst werden.*

Beweis (Skizze): eine Veranschaulichung des im Folgenden skizzierten Beweises ist in Grafik 3.6 zu erkennen. Konkret setze man

$$E(H^{(i-1)}, M^{(i)} \oplus H^{(i-1)}) := f_1(M^{(i)}, f_0(M^{(i)}, H^{(i-1)}))$$

im PGV6 Schema, da $f_0(M^{(i)}, H^{(i-1)})$ die Operation $M^{(i)} \oplus H^{(i-1)}$ generalisiert. In diesem Sinne kann nun BMW als verallgemeinertes PGV6 Schema aufgefasst werden, sodass gilt

$$H^{(i)} = f_2(M^{(i)}, H^{(i-1)}, E(H^{(i-1)}, M^{(i)} \oplus H^{(i-1)})),$$

was die Behauptung zeigt. □

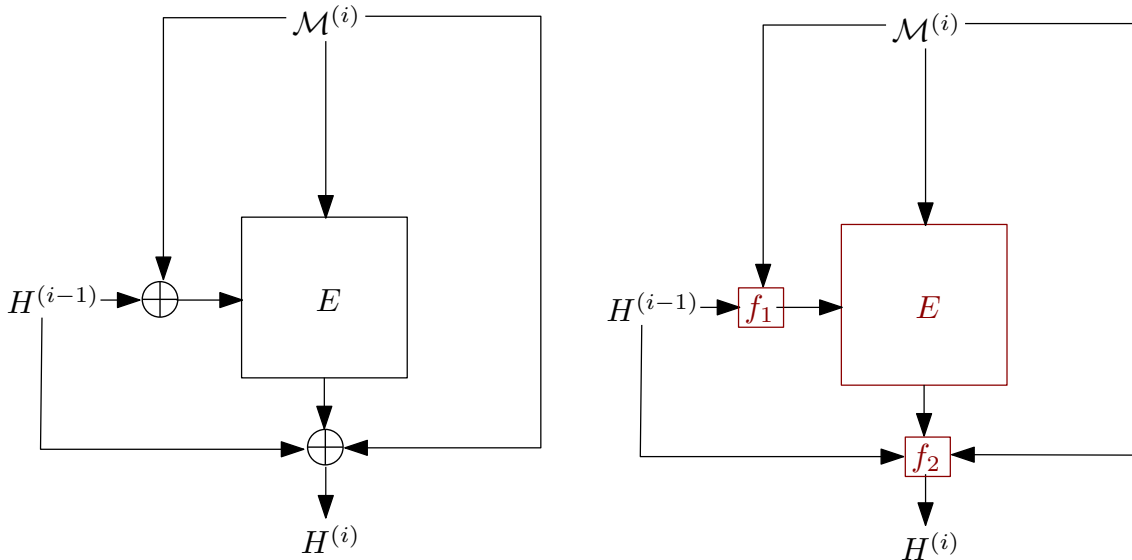
Fig.: PGV6 scheme (left), BMW hash function (right).

Abbildung 3.6: BMW als PGV6 Schema aufgefasst.

3.3 Sicherheitsannahmen

Aufbauend auf den Überlegungen aus Abschnitt 3.1 und 3.2 können wir nun die folgenden Sicherheitsbehauptungen formulieren, welche im Wesentlichen [1] entnommen sind.

Sicherheitsbehauptung 1 *Es ist praktisch nicht machbar, Kollisionen, Urbilder und zweite Urbilder zu finden.*

Begründung:

1. Das PGV6 Schema ist resistent gegen das Auffinden zweiter Urbilder und Kollisionen [5]; zudem sind f_0, f_1, f_2 nicht-lineare Funktionen.
2. Die Kompression in der Finalisation (d. h. die Funktion f_2) wendet eine zusätzliche, robuste Einwegfunktion auf den Intermediate Hash an.
3. Es ist äußerst schwer, alle drei Argumente der Funktion f_2 so zu ändern, dass sich diese aufheben oder zu kontrollierten Änderungen im Hash-Digest führen.

□

Sicherheitsbehauptung 2 *Die Approximation der Additionen und Subtraktionen in \mathbb{Z}_{2^w} durch XOR-Operationen erfordert einen immensen Berechnungsaufwand.*

Begründung: Das Problem besteht grundlegend in der Approximation von $+$ und $-$ modulo 2^n durch XORs. Die Berechnung von differentiellen Eigenschaften der Addition

modulo 2^n für zwei Variablen ist zwar machbar [6], jedoch nutzt der BMW-Algorithmus ein recht komplexes System von Additionen und Subtraktionen modulo 2^n in weit mehr als nur zwei Variablen. Insbesondere wurde gezeigt [7], dass ein Algorithmus, welcher Gleichungen dieser Gestalt löst, exponentielle Komplexität haben muss (konkret hat dieser Komplexität $O(2^{bk})$, wobei b die Bitlänge der Variablen und k die Anzahl der Gleichungen ist). \square

Sicherheitsbehauptung 3 *Differentielle Kryptoanalyse ist nicht effizient durchführbar.*

Begründung: Die differentielle Kryptoanalyse ist aus drei Gründen höchst wahrscheinlich nicht effizient durchführbar. Zum einen hat das Double-Pipe-Design den Effekt, dass ein Gegner gegenüber dem gewöhnlichen Merkle-Damgård-Design die doppelte Anzahl an Variablen im Differentialpfad benötigt. Genauso erschwert die hohe Anzahl an Permutationen (siehe Abschnitt 3.1) die differentielle Analyse. Nicht zuletzt werden einzelne Bits innerhalb der Kompressionsfunktion von BMW stark vermischt (Diffusion), wie Lemma 6 in [1] zeigt:

Lemma 6, [1]. *Jede 1-Bit-Differenz im Vektor $W^{(i)}$ nach Schritt 1 und Schritt 2 der Funktion f_0 verteilt sich in 5 Wörter des Vektors Q_a und jeder Unterschied in diesen 5 Wörtern hat mindestens ein oder zwei Bit (bzw. 3 oder 4 Bit) Unterschied.* \square

3.4 Weitere Eigenschaften von BMW

3.4.1 Bekannte Schwächen

Im Jahr 2008 fand S. Thomsen einen Fast-Kollisionsangriff [8] auf die Runde 1 Kompressionsfunktion von BMW. Die in dieser Veröffentlichung benutzte Schwäche in der Funktion $\text{AddElement}(j)$ wurde in Runde 2 von den Autoren von Blue Midnight Wish ausgebessert. Die alte Version aus Runde 1 hatte nämlich die Gestalt

$$\text{AddElement}(j) = M_j^{(i)} + M_{j+3}^{(i)} - M_{j+10}^{(i)} + K_{j+16}.$$

Im Runde 2 Algorithmus von BMW wurde diese nun zu

$$\begin{aligned} \text{AddElement}(j) = & [\text{ROTL}^{(j+1)}(M_j^{(i)}) + \text{ROTL}^{(j+4)}(M_{j+3}^{(i)}) \\ & - \text{ROTL}^{(j+11)}(M_{j+10}^{(i)}) + K_{j+16}] \oplus H_{j+7}^{(i)} \end{aligned}$$

modifiziert, in der Hoffnung, dass die Schwächen des Runde 1 Algorithmus damit beseitigt seien.

3.4.2 Kryptographische Eigenschaften

Die Autoren vermuten in [1], dass der BMW Algorithmus die folgenden kryptographischen Eigenschaften hat.

- $O(2^{n/2})$ Hashberechnungen nötig, um Kollisionen zu finden.
- $O(2^n)$ Hashberechnungen nötig, um Urbilder zu finden.
- $O(2^{n-k})$ Hashberechnungen nötig, um zweite Urbilder zu finden ($|\mathcal{M}| < 2^k$).

In Grafik 3.7 finden sich noch einmal im Überblick die kryptographischen Eigenschaften der einzelnen Varianten von BMW.

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

Abbildung 3.7: Kryptographische Eigenschaften von BMW.

3.4.3 Geschwindigkeit und Hardwareabhängigkeit

Besonders hervorzuheben ist die Hardwareunabhängigkeit von Blue Midnight Wish, zusammen mit seiner außerordentlichen Geschwindigkeit.

Konkret ist der BMW Algorithmus unter den nachfolgend aufgeführten Voraussetzungen einer der schnellsten Kandidaten im Pool der Runde 2 Kandidaten des SHA-3 Wettbewerbes. Alle im Folgenden aufgeführten Angaben beziehen sich auf die mit der BMW Dokumentation mitgelieferte C-Implementation (wobei jeweils eine für die 32-Bit und eine für die 64-Bit Variante existiert). In der Tat verhält es sich so, dass BMW auf der 64-Bit NIST Referenzplattform

- $n = 224, 256$: 7.50 cycles/byte bzw.
- $n = 384, 512$: 3.90 cycles/byte

benötigt (bei einer Länge des Hash-Wertes von $\approx 10^5$).

Der Speicherverbrauch liegt ebenfalls weit unter dem der weiteren 13 Kandidaten: Konkret benötigt BMW für einen Block $M^{(i)}$ nur 264 Bytes ($n = 224, 256$) bzw. 528 Bytes ($n = 384, 512$) im Hauptspeicher.

Demzufolge ist Blue Midnight Wish einer der schnellsten und Speichereffizientesten Algorithmen im Pool der Runde 2 Kandidaten.

A. Anhang

Die explizite Darstellung der Funktion f_0 aus dem BMW Algorithmus ist wie folgt gegeben.

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

Abbildung 3.8: Die Funktion f_0 . (Tabelle aus [2, S. 19].)

Literaturverzeichnis

- [1] Offizielle Homepage des NIST,
<http://csrc.nist.gov/groups/ST/hash/sha-3/index.html>.
- [2] Originale Dokumentation des BMW-Algorithmus,
http://people.item.ntnu.no/~danilog/Hash/BMW-SecondRound/Supporting_Documentation/BlueMidnightWishDocumentation.pdf.
- [3] A. Joux, Multicollisions in iterated hash functions - application to cascaded constructions, *Crypto 04*, LNCS 3152, pp. 306 - 316.
- [4] S. Lucks, A Failure Friendly Design Principle for Hash Functions, *Lecture Notes in Computer Science*, 2005, Volume 3788/2005, pp. 474-494, DOI 10.1007/11593447_26.
- [5] R. Govaerts B. Preneel and J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *Proceedings of CRYPTO 1993*, volume 773 of LNCS, pages 368–378, 1994.
- [6] H. Lipmaa and S. Moriai. Efficient algorithms for computing differential properties of addition. In *Proceedings of FSE 2001*, volume 2355 of LNCS, pages 336–350, 2002.
- [7] S. Paul and B. Preneel. Solving systems of differential equations of addition. In *Proceedings of ACISP 2005*, volume 3574 of LNCS, pages 75–88, 2005.
- [8] Søren S. Thomsen. A near-collision attack on the Blue Midnight Wish compression function. Online paper at <http://www2.mat.dtu.dk/people/S.Thomsen/bmw/nc-compress.pdf>, Version 2.0, 2008.