Blue Midnight Wish (BMW)

A SHA-3 Hash Competition Candidate

Daniel Schliebner

29. Juli 2010

Overview

About The Algorithm

- **Authors**: Danilo Gligoroski, Vlastimil Klima, Svein Johan Knapskog.
- **Classification**:
  - Merkle-Damgård construction,
  - wide-pipe hash.
- **Variants**: BMW224, BMW256, BMW384, BMW512.

About The Algorithm

- **Authors**: Danilo Gligoroski, Vlastimil Klima, Svein Johan Knapskog.
- **Classification**:
  - Merkle-Damgård construction,
  - wide-pipe hash.
- **Variants**: BMW224, BMW256, BMW384, BMW512.

About The Algorithm

- **Authors**: Danilo Gligoroski, Vlastimil Klima, Svein Johan Knapskog.
- **Classification**:
  - Merkle-Damgård construction,
  - wide-pipe hash.
- **Variants**: BMW224, BMW256, BMW384, BMW512.

About The Algorithm

- **Authors**: Danilo Gligoroski, Vlastimil Klima, Svein Johan Knapskog.
- **Classification**:
  - Merkle-Damgård construction,
  - wide-pipe hash.
- **Variants**: BMW224, BMW256, BMW384, BMW512.

The Child Gets A Name

- First working name: **Blue Wish**.
- Problem: Blue Wish is a reg. trademark for towels.
- A version that was produced in the **midnight** had the best characteristics. This led to the final name.

The Child Gets A Name

- First working name: **Blue Wish**.
- Problem: Blue Wish is a reg. trademark for towels.
- A version that was produced in the **midnight** had the best characteristics. This led to the final name.

The Child Gets A Name

- First working name: **Blue Wish**.
- Problem: Blue Wish is a reg. trademark for towels.
- A version that was produced in the **midnight** had the best characteristics. This led to the final name.

Notation

Definition

To this end let

- $\mathcal{M}$ be the message to be hashed of length ℓ bits;
- n bits be the length of the hash output.
- m the block size used inside the Merkle-Damgård construction.
- N the number of blocks in the padded message.
- $\mathcal{M}^{(i)}$ the i-th m-bit block of $\mathcal{M}$.
- $\mathcal{M}_j^{(i)}$ the j-th word of the i-th message block.

Notation (cont.)

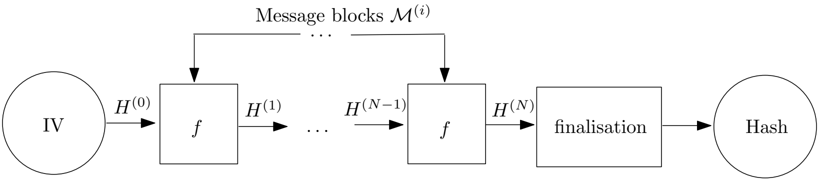
Definition

We write

- $\mathrm{SHL}^r(x)$ for a shift left operation,
- $\mathrm{SHR}^r(x)$ for a shift right operation,
- $\mathrm{ROTL}^r(x)$ for a circular left shift operation,
- $+$ and $-$ for addition/substraction modulo 2^{32}/2^{64},
- $\oplus$ for a bitwise logic word XOR-operation,

where x is in general a 32- or 64-bit word.

Merkle-Damgård Construction



Wide Pipe Hash

- Joux [Joux04] found a generic multi-collision attack against iterated hash functions.
- K-collisions in time $O(\log(K) \cdot 2^{n/2})$ (instead of $\Omega(2^{(K-1)n/K})$):
- Idea: for $K = 2^N$ compute N **local collisions** $\mathcal{M}_0^{(i)} = M_1^{(i)}$ with $f(H^{(i-1)}, \mathcal{M}_0^{(i)}) = f(H^{(i-1)}, \mathcal{M}_1^{(i)})$.
- Thus, Lucks [SLuck04] proposes to „widen“ the internal pipe.

Wide Pipe Hash

- JOUX [Joux04] found a generic multi-collision attack against iterated hash functions.
- K-collisions in time $O(\log(K) \cdot 2^{n/2})$ (instead of $\Omega(2^{(K-1)n/K})$):
- Idea: for $K = 2^N$ compute N **local collisions** $\mathcal{M}_0^{(i)} = M_1^{(i)}$ with $f(H^{(i-1)}, \mathcal{M}_0^{(i)}) = f(H^{(i-1)}, \mathcal{M}_1^{(i)})$.
- Thus, LUCKS [SLuck04] proposes to „widen“ the internal pipe.

Wide Pipe Hash

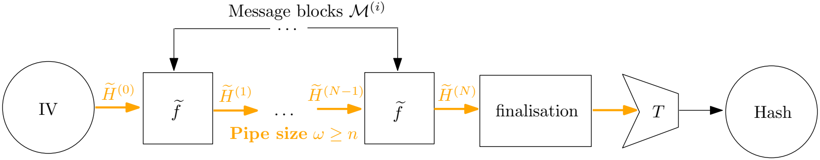
- JOUX [Joux04] found a generic multi-collision attack against iterated hash functions.
- K-collisions in time $O(\log(K) \cdot 2^{n/2})$ (instead of $\Omega(2^{(K-1)n/K})$):
- Idea: for $K = 2^N$ compute N **local collisions** $\mathcal{M}_0^{(i)} = M_1^{(i)}$ with $f(H^{(i-1)}, \mathcal{M}_0^{(i)}) = f(H^{(i-1)}, \mathcal{M}_1^{(i)})$.
- Thus, LUCKS [SLuck04] proposes to „widen“ the internal pipe.

Wide Pipe Hash

- JOUX [Joux04] found a generic multi-collision attack against iterated hash functions.
- K-collisions in time $O(\log(K) \cdot 2^{n/2})$ (instead of $\Omega(2^{(K-1)n/K})$):
- Idea: for $K = 2^N$ compute N **local collisions** $\mathcal{M}_0^{(i)} = M_1^{(i)}$ with $f(H^{(i-1)}, \mathcal{M}_0^{(i)}) = f(H^{(i-1)}, \mathcal{M}_1^{(i)})$.
- Thus, LUCKS [SLuck04] proposes to „widen“ the internal pipe.

Wide Pipe Hash (cont.)

- Extend internal pipe in MD from n bit to $\omega \geq 2n$ bit.



- Use two compression functions $\widetilde{f}$ and T.
- $\widetilde{H}^{(N)}$ is called the *intermediate hash*.

Wide Pipe Hash (cont.)

LUCKS [SLuck04] proved:

Theorem

To ensure that a wide-pipe iterated hash with an internal pipe size ω is (asymptotically) as secure against multi-collision attacks as an ideal hash, $\omega \geq 2n$ is sufficient in the random oracle model.

and even more:

Theorem

A wide-pipe iterated hash with an internal pipe size $\omega \geq 2n$ is secure against all generic attacks (i.e. they are asymptotically as secure as an ideal hash).

Wide Pipe Hash (cont.)

LUCKS [SLuck04] proved:

Theorem

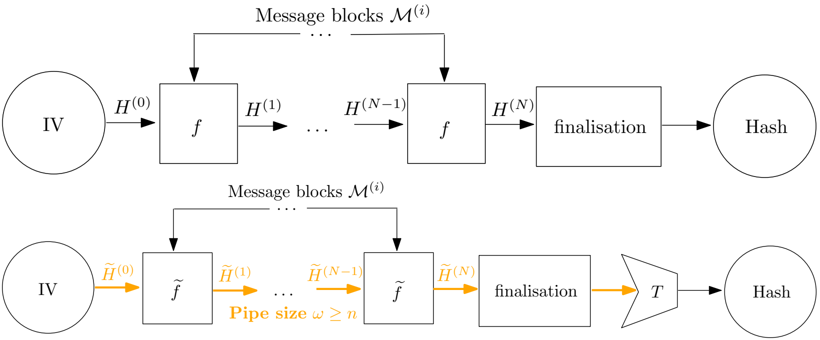
To ensure that a wide-pipe iterated hash with an internal pipe size ω is (asymptotically) as secure against multi-collision attacks as an ideal hash, $\omega \geq 2n$ is sufficient in the random oracle model.

and even more:

Theorem

A wide-pipe iterated hash with an internal pipe size $\omega \geq 2n$ is secure against all generic attacks (i.e. they are asymptotically as secure as an ideal hash).

Wide Pipe Hash (cont.)



Overview

Variants

All BMW-variants for $n \in \{224, 256, 384, 512\}$:

Algorithm abbrev.	**Message size ℓ**	**Block size m**	**Word size w**	**Digest size 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

Note:
BMW uses little-endian at byte-level and big-endian at bit-level.

Blue Midnight Wish

Input: message $\mathcal{M}$ with $|\mathcal{M}| = \ell$, hash size n.

1. **Preprocessing**
   (a) Pad the message $\mathcal{M}$.
   (b) Parse the padded message into N, m-bit message blocks $M^{(1)}, \ldots, M^{(N)}$.
   (c) Set initial value $H^{(0)}$ (m-bit).
2. **Hash computation**
   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}) $(= H_8^N || \ldots || H_{15}^N)$.

Blue Midnight Wish

Input: message $\mathcal{M}$ with $|\mathcal{M}| = \ell$, hash size n.

1. **Preprocessing**
   (a) Pad the message $\mathcal{M}$.
   (b) Parse the padded message into N, m-bit message blocks $M^{(1)}, \dots, M^{(N)}$.
   (c) Set initial value $H^{(0)}$ (m-bit).
2. **Hash computation**
   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**
   $Q_a^{\mathrm{final}} = f_0(H^N, \mathrm{F\_CONST});$
   $Q_b^{\mathrm{final}} = f_1(H^N, \mathrm{F\_CONST}, Q_a^{\mathrm{final}});$
   $H^{\mathrm{final}} = f_2(H^N, Q_a^{\mathrm{final}}, Q_b^{\mathrm{final}});$
4. **Output:** n-LSB(H^{final}) $(= H_8^N || \dots || H_{15}^N)$.

Blue Midnight Wish

Input: message $\mathcal{M}$ with $|\mathcal{M}| = \ell$, hash size n.

1. **Preprocessing**
   (a) Pad the message $\mathcal{M}$.
   (b) Parse the padded message into N, m-bit message blocks $M^{(1)}, \dots, M^{(N)}$.
   (c) Set initial value $H^{(0)}$ (m-bit).
2. **Hash computation**
   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}) $(= H_8^N || \dots || H_{15}^N)$.

Blue Midnight Wish

Input: message $\mathcal{M}$ with $|\mathcal{M}| = \ell$, hash size n.

1. **Preprocessing**
   (a) Pad the message $\mathcal{M}$.
   (b) Parse the padded message into N, m-bit message blocks $M^{(1)}, \ldots, M^{(N)}$.
   (c) Set initial value $H^{(0)}$ (m-bit).
2. **Hash computation**
   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**
   $$Q_a^{\mathrm{final}} = f_0(H^N, \mathrm{F\_CONST});$$
   $$Q_b^{\mathrm{final}} = f_1(H^N, \mathrm{F\_CONST}, Q_a^{\mathrm{final}});$$
   $$H^{\mathrm{final}} = f_2(H^N, Q_a^{\mathrm{final}}, Q_b^{\mathrm{final}});$$
4. **Output:** n-LSB(H^{final}) $(= H_8^N || \ldots || H_{15}^N)$.

Overview

1. Preprocessing

- **ad (a)**: pad message to

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

where $\ell + k + 1 \equiv_{512} 448$ (or $\ell + k + 1 \equiv_{1024} 960$).

- **ad (b)**: parse padded message into N, m-bit blocks $M^{(i)}$, $i = 1, \ldots, N$.
- **ad (c)**: m-bit initial value $H^{(0)}$ depends on $n \in \{224, 256, 384, 512\}$.
  - $n = 224$: parse $0\text{x}00 \ldots 0\text{x}3F$ to sixteen 32-bit words.
  - $n = 256$: parse $0\text{x}40 \ldots 0\text{x}7F$ to sixteen 32-bit words.
  - $n = 384$: parse $0\text{x}00 \ldots 0\text{x}7F$ to sixteen 64-bit words.
  - $n = 512$: parse $0\text{x}80 \ldots 0\text{x}FF$ to sixteen 64-bit words.

1. Preprocessing

- **ad (a)**: pad message to

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

where $\ell + k + 1 \equiv_{512} 448$ (or $\ell + k + 1 \equiv_{1024} 960$).

- **ad (b)**: parse padded message into N, m-bit blocks $M^{(i)}$, $i = 1, \ldots, N$.
- **ad (c)**: m-bit initial value $H^{(0)}$ depends on $n \in \{224, 256, 384, 512\}$.
  - $n = 224$: parse $0x00 \ldots 0x3F$ to sixteen 32-bit words.
  - $n = 256$: parse $0x40 \ldots 0x7F$ to sixteen 32-bit words.
  - $n = 384$: parse $0x00 \ldots 0x7F$ to sixteen 64-bit words.
  - $n = 512$: parse $0x80 \ldots 0xFF$ to sixteen 64-bit words.

1. Preprocessing

- **ad (a)**: pad message to

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

  where $\ell + k + 1 \equiv_{512} 448$ (or $\ell + k + 1 \equiv_{1024} 960$).
- **ad (b)**: parse padded message into N, m-bit blocks $M^{(i)}$, $i = 1, \ldots, N$.
- **ad (c)**: m-bit initial value $H^{(0)}$ depends on $n \in \{224, 256, 384, 512\}$.
  - $n = 224$: parse $0x00 \ldots 0x3F$ to sixteen 32-bit words.
  - $n = 256$: parse $0x40 \ldots 0x7F$ to sixteen 32-bit words.
  - $n = 384$: parse $0x00 \ldots 0x7F$ to sixteen 64-bit words.
  - $n = 512$: parse $0x80 \ldots 0xFF$ to sixteen 64-bit words.

Overview

2. Hash computation

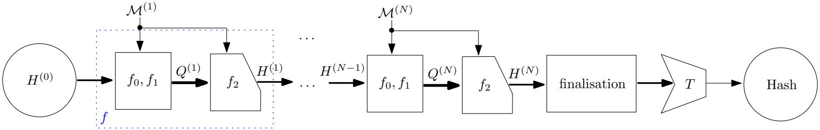


Fig.: The BMW algorithm

- Compression function splits into:

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

2. Hash computation (cont.)

Definition of f_0

- Define $f_0(M^{(i)}, H^{(i-1)}) := Q_a^{(i)}$, where
$$Q_a^{(i)} := A_2(A_1(M^{(i)} \oplus H^{(i-1)})) + \mathrm{ROTL}^1(H^{(i-1)})$$
- $A_1 \in \{0, 1, -1\}^{16\times 16}$ is obtained by a matrix $A_1' \in \mathbb{F}_2^{16\times 16}$ by randomly turning some values '1' to '-1' so that $\det A_1 \in \mathbb{Z}_{2^w}^\times$.
- Let $W^{(i)} := A_1(M^{(i)} \oplus H^{(i-1)})$. Then
$$A_2(W_j^{(i)}) := s_{j \bmod 5}(W_j^{(i)}) + H_{(j+1) \bmod 16}^{(i-1)}$$
- s_0, s_1, s_2, s_3, s_4 are transformations based on $\mathrm{ROTL}, \mathrm{SHL}, \mathrm{SHR}$ (see handout).

2. Hash computation (cont.)

Definition of f_0

- Define $f_0(M^{(i)}, H^{(i-1)}) := Q_a^{(i)}$, where
$$Q_a^{(i)} := A_2(A_1(M^{(i)} \oplus H^{(i-1)})) + \mathrm{ROTL}^1(H^{(i-1)})$$
- $A_1 \in \{0, 1, -1\}^{16\times 16}$ is obtained by a matrix $A_1' \in \mathbb{F}_2^{16\times 16}$ by randomly turning some values '1' to '-1' so that $\det A_1 \in \mathbb{Z}_{2w}^{\times}$.
- Let $W^{(i)} := A_1(M^{(i)} \oplus H^{(i-1)})$. Then
$$A_2(W_j^{(i)}) := s_{j \bmod 5}(W_j^{(i)}) + H_{(j+1) \bmod 16}^{(i-1)}$$
- s_0, s_1, s_2, s_3, s_4 are transformations based on $\mathrm{ROTL}, \mathrm{SHL}, \mathrm{SHR}$ (see handout).

2. Hash computation (cont.)

Definition of f_0

- Define $f_0(M^{(i)}, H^{(i-1)}) := Q_a^{(i)}$, where
$$Q_a^{(i)} := A_2(A_1(M^{(i)} \oplus H^{(i-1)})) + \mathrm{ROTL}^1(H^{(i-1)})$$
- $A_1 \in \{0, 1, -1\}^{16\times 16}$ is obtained by a matrix $A_1' \in \mathbb{F}_2^{16\times 16}$ by randomly turning some values '1' to '-1' so that $\det A_1 \in \mathbb{Z}_{2^w}^{\times}$.
- Let $W^{(i)} := A_1(M^{(i)} \oplus H^{(i-1)})$. Then
$$A_2(W_j^{(i)}) := s_{j \bmod 5}(W_j^{(i)}) + H^{(i-1)}_{(j+1) \bmod 16}$$
- s_0, s_1, s_2, s_3, s_4 are transformations based on $\mathrm{ROTL}, \mathrm{SHL}, \mathrm{SHR}$ (see handout).

2. Hash computation (cont.)

Definition of f_0

- Define $f_0(M^{(i)}, H^{(i-1)}) := Q_a^{(i)}$, where
$$Q_a^{(i)} := A_2(A_1(M^{(i)} \oplus H^{(i-1)})) + \mathrm{ROTL}^1(H^{(i-1)})$$
- $A_1 \in \{0, 1, -1\}^{16\times 16}$ is obtained by a matrix $A'_1 \in \mathbb{F}_2^{16\times 16}$ by randomly turning some values '1' to '-1' so that $\det A_1 \in \mathbb{Z}_{2^w}^{\times}$.
- Let $W^{(i)} := A_1(M^{(i)} \oplus H^{(i-1)})$. Then
$$A_2(W_j^{(i)}) := s_{j \bmod 5}(W_j^{(i)}) + H_{(j+1) \bmod 16}^{(i-1)}$$
- s_0, s_1, s_2, s_3, s_4 are transformations based on $\mathrm{ROTL}, \mathrm{SHL}, \mathrm{SHR}$ (see handout).

2. Hash computation (cont.)

Definition of f_0

1. Bijective transform of $M^{(i)} \oplus H^{(i-1)}$:

$$
\begin{aligned}
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)})
\end{aligned}
$$

2. Further bijective transform of $W_j^{(i)}, j = 0, \ldots, 15$:

$$
\begin{aligned}
&Q_0^{(i)} = s_0(W_0^{(i)}) + H_1^{(i-1)}; && Q_1^{(i)} = s_1(W_1^{(i)}) + H_2^{(i-1)}; && Q_2^{(i)} = s_2(W_2^{(i)}) + H_3^{(i-1)}; && Q_3^{(i)} = s_3(W_3^{(i)}) + H_4^{(i-1)}; \\
&Q_4^{(i)} = s_4(W_4^{(i)}) + H_5^{(i-1)}; && Q_5^{(i)} = s_0(W_5^{(i)}) + H_6^{(i-1)}; && Q_6^{(i)} = s_1(W_6^{(i)}) + H_7^{(i-1)}; && Q_7^{(i)} = s_2(W_7^{(i)}) + H_8^{(i-1)}; \\
&Q_8^{(i)} = s_3(W_8^{(i)}) + H_9^{(i-1)}; && Q_9^{(i)} = s_4(W_9^{(i)}) + H_{10}^{(i-1)}; && Q_{10}^{(i)} = s_0(W_{10}^{(i)}) + H_{11}^{(i-1)}; && Q_{11}^{(i)} = s_1(W_{11}^{(i)}) + H_{12}^{(i-1)}; \\
&Q_{12}^{(i)} = s_2(W_{12}^{(i)}) + H_{13}^{(i-1)}; && Q_{13}^{(i)} = s_3(W_{13}^{(i)}) + H_{14}^{(i-1)}; && Q_{14}^{(i)} = s_4(W_{14}^{(i)}) + H_{15}^{(i-1)}; && Q_{15}^{(i)} = s_0(W_{15}^{(i)}) + H_0^{(i-1)};
\end{aligned}
$$

2. Hash computation (cont.)

Definition of f_1

- $f_1(M^{(i)}, H^{(i-1)}, Q_a^{(i)}) := Q_b^{(i)}$ with $Q_b^{(i)} = (Q_{16}^{(i)}, \ldots, Q_{31}^{(i)})$
- Compute:
  for $ii = 0$ **to** ExpandRounds$_1$ - 1
  $Q_{ii+16}^{(i)} = \text{expand}_1(ii + 16);$
  for $ii =$ExpandRounds$_1$ **to** ExpandRounds$_1$ + ExpandRounds$_2$ - 1
  $Q_{ii+16}^{(i)} = \text{expand}_2(ii + 16);$
- expand_1 uses s_k, expand_2 more simple rotations r_k (see handout).
- ExpandRounds$_1 = 2$, ExpandRounds$_2 = 14$ is default (best secure-speed ratio).

2. Hash computation (cont.)

Definition of f_1

- $f_1(M^{(i)}, H^{(i-1)}, Q_a^{(i)}) := Q_b^{(i)}$ with $Q_b^{(i)} = (Q_{16}^{(i)}, \dots, Q_{31}^{(i)})$
- Compute:
  for $ii = 0$ **to** ExpandRounds$_1$ - 1
  $Q_{ii+16}^{(i)} = \text{expand}_1(ii + 16);$
  for $ii =$ExpandRounds$_1$ **to** ExpandRounds$_1$ + ExpandRounds$_2$ - 1
  $Q_{ii+16}^{(i)} = \text{expand}_2(ii + 16);$
- expand_1 uses s_k, expand_2 more simple rotations r_k (see handout).
- ExpandRounds$_1$ $= 2$, ExpandRounds$_2$ $= 14$ is default (best secure-speed ratio).

2. Hash computation (cont.)

Definition of f_1

- $f_1(M^{(i)}, H^{(i-1)}, Q_a^{(i)}) := Q_b^{(i)}$ with $Q_b^{(i)} = (Q_{16}^{(i)}, \ldots, Q_{31}^{(i)})$
- Compute:
  for $ii = 0$ **to** ExpandRounds$_1$ - 1
  $Q_{ii+16}^{(i)} = \mathrm{expand}_1(ii + 16);$
  for $ii =$ExpandRounds$_1$ **to** ExpandRounds$_1$ + ExpandRounds$_2$ - 1
  $Q_{ii+16}^{(i)} = \mathrm{expand}_2(ii + 16);$
- expand_1 uses s_k, expand_2 more simple rotations r_k (see handout).
- ExpandRounds$_1 = 2$, ExpandRounds$_2 = 14$ is default (best secure-speed ratio).

2. Hash computation (cont.)

Definition of f_1

- $f_1(M^{(i)}, H^{(i-1)}, Q_a^{(i)}) := Q_b^{(i)}$ with $Q_b^{(i)} = (Q_{16}^{(i)}, \dots, Q_{31}^{(i)})$
- Compute:
  for $ii = 0$ **to** ExpandRounds$_1$ - 1
  $Q_{ii+16}^{(i)} = \mathrm{expand}_1(ii + 16)$;
  for $ii =$ExpandRounds$_1$ **to** ExpandRounds$_1$ + ExpandRounds$_2$ - 1
  $Q_{ii+16}^{(i)} = \mathrm{expand}_2(ii + 16)$;
- expand_1 uses s_k, expand_2 more simple rotations r_k (see handout).
- ExpandRounds$_1$ $= 2$, ExpandRounds$_2$ $= 14$ is default (best secure-speed ratio).

2. Hash computation (cont.)

Definition of f_2

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

2. Hash computation (cont.)

Definition of f_2

1. Compute the cumulative temporary variables XL and XH.

$$
\begin{array}{rcccccccc}
XL = & & & Q_{16}^{(i)} & \oplus & Q_{17}^{(i)} & \oplus \; \ldots \; \oplus & Q_{23}^{(i)} \\
XH = & XL & \oplus & Q_{24}^{(i)} & \oplus & Q_{25}^{(i)} & \oplus \; \ldots \; \oplus & Q_{31}^{(i)}
\end{array}
$$

2. Compute the new double pipe $H^{(i)}$:

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

Overview

3. Finalisation

For $n \in \{224, 256\}$ set:
$\mathrm{F\_CONST} = (0\mathrm{x}aaaaaaa0, \ldots, 0\mathrm{x}aaaaaaaf)$,

For $n \in \{384, 512\}$ set:
$\mathrm{F\_CONST} =$
$(0\mathrm{x}aaaaaaaaaaaaaaa0, \ldots, 0\mathrm{x}aaaaaaaaaaaaaaaf)$.

Overview

Cornerstones

- Double pipe design.
- Heavily usage of permutations/bijections.
- Nonlinear expressions inside the f_i functions, $i = 0, 1, 2$.

Double pipe design

Ensures **provable** security against generic attacks (i.e. attacks that are still applicatable if one replaces the compressions function against an oracle).
(Theorem of LUCKS, [SLuck04]).

Permutations and bijections

The authors proved:

Theorem

1. *When* $H^{(i-1)}$ *is fixed,* $f_0(\cdot, H^{(i-1)})$ *is a bijection.*
2. *For given* $H^{(i-1)}$ *the function* $f_1(\cdot, H^{(i-1)}, \cdot)$ *is a multipermutation.*
3. *When* $Q_b^{(i)}$ *and* $M^{(i)}$ *are fixed,* $f_2(M^{(i)}, \cdot, Q_b^{(i)})$ *is a bijection.*
4. *When* $Q_b^{(i)}$ *and* $Q_a^{(i)}$ *are fixed,* $f_2(\cdot, Q_a^{(i)}, Q_b^{(i)})$ *is a bijection.*

Proof: [BMW09, p. 35] □

Nonlinear expressions inside the f_2 function

- f_i nonlinear since $+$ and $-$ in $\mathbb{Z}_{2^{32}}$ (or $\mathbb{Z}_{2^{64}}$) are nonlinear operations in $\mathrm{GF}(2^{32})$ (or $\mathrm{GF}(2^{64})$).
- Preneel, Govaerts, and Vandewalle have located 12 secure schemes for constructing hash functions from block ciphers (namely PGV1-PGV12).
- PGV6: $H^{(i)} = E(H^{(i-1)}, M^{(i)} \oplus H^{(i-1)}) \oplus M^{(i)} \oplus H^{(i-1)}$.

Nonlinear expressions inside the f_2 function

- f_i nonlinear since $+$ and $-$ in $\mathbb{Z}_{2^{32}}$ (or $\mathbb{Z}_{2^{64}}$) are nonlinear operations in $\mathrm{GF}(2^{32})$ (or $\mathrm{GF}(2^{64})$).
- Preneel, Govaerts, and Vandewalle have located 12 secure schemes for constructing hash functions from block ciphers (namely PGV1-PGV12).
- PGV6: $H^{(i)} = E(H^{(i-1)}, M^{(i)} \oplus H^{(i-1)}) \oplus M^{(i)} \oplus H^{(i-1)}$.

Nonlinear expressions inside the f_2 function

- f_i nonlinear since $+$ and $-$ in $\mathbb{Z}_{2^{32}}$ (or $\mathbb{Z}_{2^{64}}$) are nonlinear operations in $\mathrm{GF}(2^{32})$ (or $\mathrm{GF}(2^{64})$).
- Preneel, Govaerts, and Vandewalle have located 12 secure schemes for constructing hash functions from block ciphers (namely PGV1-PGV12).
- PGV6: $H^{(i)} = E(H^{(i-1)}, M^{(i)} \oplus H^{(i-1)}) \oplus M^{(i)} \oplus H^{(i-1)}$.

Nonlinear expressions inside the f_2 function (cont.)

Theorem

BMW hash function can be expressed as a generalized PGV6 scheme.

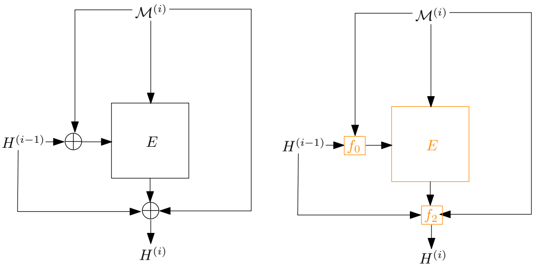


Fig.: PGV6 scheme (left), BMW hash function (right).

Nonlinear expressions inside the f_2 function (cont.)

Theorem

BMW hash function can be expressed as as generalized PGV6 scheme.

Proof (scetch):

- Set $E(H^{(i-1)}, M^{(i)} \oplus H^{(i-1)}) := f_1(M^{(i)}, f_0(M^{(i)}, H^{(i-1)}))$, since $f_0(M^{(i)}, H^{(i-1)})$ generalizes $M^{(i)} \oplus H^{(i-1)}$.
- Thus, BMW can be represented as generalized PGV6 scheme such that $H^{(i)} = f_2(M^{(i)}, H^{(i-1)}, E(H^{(i-1)}, M^{(i)} \oplus H^{(i-1)}))$.

Nonlinear expressions inside the f_2 function (cont.)

Theorem

BMW hash function can be expressed as as generalized PGV6 scheme.

Proof (scetch):

- Set $E(H^{(i-1)}, M^{(i)} \oplus H^{(i-1)}) := f_1(M^{(i)}, f_0(M^{(i)}, H^{(i-1)}))$, since $f_0(M^{(i)}, H^{(i-1)})$ generalizes $M^{(i)} \oplus H^{(i-1)}$.
- Thus, BMW can be represented as generalized PGV6 scheme such that $H^{(i)} = f_2(M^{(i)}, H^{(i-1)}, E(H^{(i-1)}, M^{(i)} \oplus H^{(i-1)}))$.

Nonlinear expressions inside the f_2 function (cont.)

Theorem

BMW hash function can be expressed as as generalized PGV6 scheme.

Proof (scetch):

- Set $E(H^{(i-1)}, M^{(i)} \oplus H^{(i-1)}) := f_1(M^{(i)}, f_0(M^{(i)}, H^{(i-1)}))$, since $f_0(M^{(i)}, H^{(i-1)})$ generalizes $M^{(i)} \oplus H^{(i-1)}$.
- Thus, BMW can be represented as generalized PGV6 scheme such that $H^{(i)} = f_2(M^{(i)}, H^{(i-1)}, E(H^{(i-1)}, M^{(i)} \oplus H^{(i-1)}))$.

$\square$

Overview

Claims

Claim

It is infeasable to find collisions, preimages and second preimages.

Reasons:

1. PGV6 scheme is second-preimage and collision resistant, moreover the functions f_0, f_1, f_2 are nonlinear.
2. Compression in finalisation (namely f_2) applies another robust one-way function on the result.
3. It is hard to change consistently all three inputs of f_2 such that they will cancel out each other or lead to controllable changes.

Claims

Claim

It is infeasable to find collisions, preimages and second preimages.

Reasons:

1. PGV6 scheme is second-preimage and collision resistant, moreover the functions f_0, f_1, f_2 are nonlinear.
2. Compression in finalisation (namely f_2) applies another robust one-way function on the result.
3. It is hard to change consistently all three inputs of f_2 such that they will cancel out each other or lead to controllable changes.

Claims

Claim

It is infeasable to find collisions, preimages and second preimages.

Reasons:

1. PGV6 scheme is second-preimage and collision resistant, moreover the functions f_0, f_1, f_2 are nonlinear.
2. Compression in finalisation (namely f_2) applies another robust one-way function on the result.
3. It is hard to change consistently all three inputs of f_2 such that they will cancel out each other or lead to controllable changes.

Claims

Claim

It is infeasable to find collisions, preimages and second preimages.

Reasons:

1. PGV6 scheme is second-preimage and collision resistant, moreover the functions f_0, f_1, f_2 are nonlinear.
2. Compression in finalisation (namely f_2) applies another robust one-way function on the result.
3. It is hard to change consistently all three inputs of f_2 such that they will cancel out each other or lead to controllable changes.

Claims (cont.)

Claim

Approximation of additions and subtractions with XOR is computationally to expensive.

Reasons:

1. **Problem**: approximate $+/-$ modulo 2^n with XORs.
2. Computing the differential properties of addition modulo 2^n for two variables is feasible (Lipmaa/Moriai, 2001).
3. **But**: BMW uses a complex system of $+/-$ with a lot more than two variables. An algorithm to solve such equations has exponential complexity, i. e. is of order $O(2^{bk})$, where b is the bit length of the variables and k is the number of equations (Paul and Preneel, 2005).

Claims (cont.)

Claim

Approximation of additions and subtractions with XOR is computationally to expensive.

Reasons:

1. **Problem**: approximate $+/-$ modulo 2^n with XORs.
2. Computing the differential properties of addition modulo 2^n for two variables is feasible (Lipmaa/Moriai, 2001).
3. **But**: BMW uses a complex system of $+/-$ with a lot more than two variables. An algorithm to solve such equations has exponential complexity, i. e. is of order $O(2^{bk})$, where b is the bit length of the variables and k is the number of equations (Paul and Preneel, 2005).

Claims (cont.)

Claim

Approximation of additions and subtractions with XOR is computationally to expensive.

Reasons:

1. **Problem**: approximate $+/-$ modulo 2^n with XORs.
2. Computing the differential properties of addition modulo 2^n for two variables is feasible (Lipmaa/Moriai, 2001).
3. **But**: BMW uses a complex system of $+/-$ with a lot more than two variables. An algorithm to solve such equations has exponential complexity, i. e. is of order $O(2^{bk})$, where b is the bit length of the variables and k is the number of equations (Paul and Preneel, 2005).

Claims (cont.)

Claim

Approximation of additions and subtractions with XOR is computationally to expensive.

Reasons:

1. **Problem**: approximate $+/-$ modulo 2^n with XORs.
2. Computing the differential properties of addition modulo 2^n for two variables is feasible (Lipmaa/Moriai, 2001).
3. **But**: BMW uses a complex system of $+/-$ with a lot more than two variables. An algorithm to solve such equations has exponential complexity, i. e. is of order $O(2^{bk})$, where b is the bit length of the variables and k is the number of equations (Paul and Preneel, 2005).

Claims (cont.)

Claim

Differential cryptanalysis is infeasable.

Reasons:

1. Double pipe design has the effect that the adversary has to use twice the number of variables in the differential path than in a single pipe.
2. Heavily usage of diffusions:
   „*Every one bit difference in the vector $W^{(i)}$ 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 minimum 1 or 2 bits difference, or minimum 3 or 4 bits difference*“ ([BMW09, Lemma 6]).
3. Heavily usage of permutations.

Claims (cont.)

Claim

Differential cryptanalysis is infeasable.

Reasons:

1. Double pipe design has the effect that the adversary has to use twice the number of variables in the differential path than in a single pipe.
2. Heavily usage of diffusions:
„*Every one bit difference in the vector $W^{(i)}$ 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 minimum 1 or 2 bits difference, or minimum 3 or 4 bits difference*“ ([BMW09, Lemma 6]).
3. Heavily usage of permutations.

Claims (cont.)

Claim

Differential cryptanalysis is infeasable.

Reasons:

1. Double pipe design has the effect that the adversary has to use twice the number of variables in the differential path than in a single pipe.
2. Heavily usage of diffusions:
„*Every one bit difference in the vector* $W^{(i)}$ *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 minimum 1 or 2 bits difference, or minimum 3 or 4 bits difference*“ ([BMW09, Lemma 6]).
3. Heavily usage of permutations.

Claims (cont.)

Claim

Differential cryptanalysis is infeasable.

Reasons:

1. Double pipe design has the effect that the adversary has to use twice the number of variables in the differential path than in a single pipe.
2. Heavily usage of diffusions:
„*Every one bit difference in the vector $W^{(i)}$ 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 minimum 1 or 2 bits difference, or minimum 3 or 4 bits difference*“ ([BMW09, Lemma 6]).
3. Heavily usage of permutations.

Overview

Vulnerabilities

- A near collision attack on the BMW compression function (by Thomsen, 2008).
- Attack at the Round 1 BMW hash function.
- Vulnerabilitiy fixed by modification of $\mathrm{AddElement}(j)$ function.
- Old version: $\mathrm{AddElement}(j) = M_j^{(i)} + M_{j+3}^{(i)} - M_{j+10}^{(i)} + K_{j+16}$.
- New version: $\mathrm{AddElement}(j) = [\mathrm{ROTL}^{(j+1)}(M_j^{(i)}) + \mathrm{ROTL}^{(j+4)}(M_{j+3}^{(i)}) - \mathrm{ROTL}^{(j+11)}(M_{j+10}^{(i)}) + K_{j+16}] \oplus H_{j+7}^{(i)}$.

Vulnerabilities

- A near collision attack on the BMW compression function (by Thomsen, 2008).
- Attack at the Round 1 BMW hash function.
- Vulnerabilitiy fixed by modification of $\mathrm{AddElement}(j)$ function.
- Old version: $\mathrm{AddElement}(j) = M_j^{(i)} + M_{j+3}^{(i)} - M_{j+10}^{(i)} + K_{j+16}$.
- New version: $\mathrm{AddElement}(j) = [\mathrm{ROTL}^{(j+1)}(M_j^{(i)}) + \mathrm{ROTL}^{(j+4)}(M_{j+3}^{(i)}) - \mathrm{ROTL}^{(j+11)}(M_{j+10}^{(i)}) + K_{j+16}] \oplus H_{j+7}^{(i)}$.

Vulnerabilities

- A near collision attack on the BMW compression function (by Thomsen, 2008).
- Attack at the Round 1 BMW hash function.
- Vulnerabilitiy fixed by modification of $\mathrm{AddElement}(j)$ function.
- Old version: $\mathrm{AddElement}(j) = M_j^{(i)} + M_{j+3}^{(i)} - M_{j+10}^{(i)} + K_{j+16}$.
- New version: $\mathrm{AddElement}(j) = [\mathrm{ROTL}^{(j+1)}(M_j^{(i)}) + \mathrm{ROTL}^{(j+4)}(M_{j+3}^{(i)}) - \mathrm{ROTL}^{(j+11)}(M_{j+10}^{(i)}) + K_{j+16}] \oplus H_{j+7}^{(i)}$.

Vulnerabilities

- A near collision attack on the BMW compression function (by Thomsen, 2008).
- Attack at the Round 1 BMW hash function.
- Vulnerabilitiy fixed by modification of $\mathrm{AddElement}(j)$ function.
- Old version:
  $\mathrm{AddElement}(j) = M_j^{(i)} + M_{j+3}^{(i)} - M_{j+10}^{(i)} + K_{j+16}$.
- New version:
  $\mathrm{AddElement}(j) = [\mathrm{ROTL}^{(j+1)}(M_j^{(i)}) + \mathrm{ROTL}^{(j+4)}(M_{j+3}^{(i)}) - \mathrm{ROTL}^{(j+11)}(M_{j+10}^{(i)}) + K_{j+16}] \oplus H_{j+7}^{(i)}$.

Cryptographic strength

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

- $O(2^{n/2})$ hash computations for finding collisions.
- $O(2^n)$ hash computations for finding preimages.
- $O(2^{n-k})$ hash computations for finding second preimages ($|\mathcal{M}| < 2^k$).

Cryptographic strength

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

- $O(2^{n/2})$ hash computations for finding collisions.
- $O(2^n)$ hash computations for finding preimages.
- $O(2^{n-k})$ hash computations for finding second preimages ($|\mathcal{M}| < 2^k$).

Speed and requirements

- BMW requires **no specific hardware**.
- On NIST reference platform (64 bit version):
  - $n = 224, 256 : 7.50$ cycles/byte
  - $n = 384, 512 : 3.90$ cycles/byte
- Memory requirements: for one block $M^{(i)}$, BMW needs 264 bytes ($n = 224, 256$) and 528 bytes ($n = 384, 512$).
- **Thus, BMW is one of the fastest algorithms under the 14 SHA-3 candidates.**

Speed and requirements

- BMW requires **no specific hardware**.
- On NIST reference platform (64 bit version):
  - $n = 224, 256 : 7.50$ cycles/byte
  - $n = 384, 512 : 3.90$ cycles/byte
- Memory requirements: for one block $M^{(i)}$, BMW needs 264 bytes ($n = 224, 256$) and 528 bytes ($n = 384, 512$).
- **Thus, BMW is one of the fastest algorithms under the 14 SHA-3 candidates.**

Speed and requirements

- BMW requires **no specific hardware**.
- On NIST reference platform (64 bit version):
  - $n = 224, 256 : 7.50$ cycles/byte
  - $n = 384, 512 : 3.90$ cycles/byte
- Memory requirements: for one block $M^{(i)}$, BMW needs 264 bytes ($n = 224, 256$) and 528 bytes ($n = 384, 512$).
- **Thus, BMW is one of the fastest algorithms under the 14 SHA-3 candidates.**

Speed and requirements

- BMW requires **no specific hardware**.
- On NIST reference platform (64 bit version):
  - $n = 224, 256 : 7.50$ cycles/byte
  - $n = 384, 512 : 3.90$ cycles/byte
- Memory requirements: for one block $M^{(i)}$, BMW needs 264 bytes ($n = 224, 256$) and 528 bytes ($n = 384, 512$).
- **Thus, BMW is one of the fastest algorithms under the 14 SHA-3 candidates.**

That's it. Thank you for your attention.

Literaturverzeichnis

- [BMW09] Cryptographic Hash Function BLUE MIDNIGHT WISH
  Danilo Gligoroski, Vlastimil Klima, Svein Johan Knapskog.
  Submission to NIST (Round 2), 2009.
- [Joux04] Multicollisions in iterated hash functions, application to cascaded constructions.
  A. Joux.
  Crypto 04, LNCS 3152, pp. 306 - 316.
- [SLuck04] Design Principles for Iterated Hash Functions.
  Stefan Lucks.
  e-Print, University of Mannheim, Germany.