Study and Analysis of BLAKE2 Hash Function

MS Project (ECS502)

Under the guidance of Dr Shashank Singh

Ajay Choudhury

Department of Electrical Engineering and Computer Science (EECS)
Indian Institute of Science Education and Research Bhopal

April 2023

Cryptographic Hash Functions

Cryptographic Hash functions are mathematical functions that take binary strings of arbitrary length as input and output strings of some fixed length (known as hash digests). Mathematically, a hash function can be represented as:

$$h : \{0,1\}^{\infty} \rightarrow \{0,1\}^{n} \tag{1}$$

where **n** is the number of bits the hash function produces as output. Some examples of commonly known hash functions are `MD5`, `SHA-I`, `SHA-256`, etc.

Cryptographic hash functions are such hash functions(many-to-one functions) for which it is computationally hard to get the two elements that map to the same element.

Features of Cryptographic Hash Functions

- **Collision Resistance:** A hash function H is said to be collision resistant if it is computationally impractical to find x and x' such that $H(x) = H(x')$.
- **Preimage Resistance:** A hash function is said to be preimage resistant if it is computationally impractical for a polynomial-time algorithm to predict the message x given it's hash y such that $H(x) = y$.
- **Second Preimage Resistance:** A hash function is said to be second preimage resistant if given a message x it is not feasible for a polynomial-time algorithm to compute y such that $H(x) = H(y)$.

BLAKE and BLAKE2 Hash Functions

- BLAKE-256 is one of the five finalists of the NIST hash function competition for SHA-III standard held in 2008.
- Similar security standards as that of SHA-III standard.
- Design and implementation are much simpler and interpretable.
- Produces hash digests of any size between 1 and 64 bytes.

- BLAKE2 is faster than the BLAKE hash function (close to MD5).
- Padding scheme in BLAKE2 is modified as compared to BLAKE.
- BLAKE2 generates hash in little-endian format, as the majority of the target platforms are little-endian in contrast to the big-endian format of BLAKE.
- In BLAKE2, the counter variable is computed in bytes instead of bits.

Constants and Predefined Objects in BLAKE-256

The 8 constant initialization vectors ($\mathtt{IV}_0, \ldots, \mathtt{IV}_7$) are used in BLAKE-256 and BLAKE2s to initialize the chain variables (h_i) and the state variables (v_i).

$\mathtt{IV}_0$ = 0x6A09E667;	$\mathtt{IV}_1$ = 0xBB67AE85;	$\mathtt{IV}_2$ = 0x3C6EF372;	$\mathtt{IV}_3$ = 0xA54FF53A;
$\mathtt{IV}_4$ = 0x510E527F;	$\mathtt{IV}_5$ = 0x9B05688C;	$\mathtt{IV}_6$ = 0x1F83D9AB;	$\mathtt{IV}_7$ = 0x5BE0CD19;

16 word constants ($k_0, \ldots, k_{15}$) are used in the core function of BLAKE-256. These are not used in BLAKE2s.

k_0 = 0x243F6A88;	k_1 = 0x85A308D3;	k_2 = 0x13198A2E;	k_3 = 0x03707344;
k_4 = 0xA4093822;	k_5 = 0x299F31D0;	k_6 = 0x082EFA98;	k_7 = 0xEC4E6C89;
k_8 = 0x452821E6;	k_9 = 0x38D01377;	k_{10} = 0xBE5466CF;	k_{11} = 0x34E90C6C;
k_{12} = 0xC0AC29B7;	k_{13} = 0xC97C50DD;	k_{14} = 0x3F84D5B5;	k_{15} = 0xB5470917;

Permutation table used for Message-block Selection

The permutation table is used to select the message blocks (m_i) and constants (k_i) in the core function, and it is the same for both `BLAKE-256` and `BLAKE2s`.

Round	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	14	10	4	8	9	15	13	6	1	12	0	2	11	7	5	3
3	11	8	12	0	5	2	15	13	10	14	3	6	7	1	9	4
4	7	9	3	1	13	12	11	14	2	6	5	10	4	0	15	8
5	9	0	5	7	2	4	10	15	14	1	11	12	6	8	3	13
6	2	12	6	10	0	11	8	3	4	13	7	5	15	14	1	9
7	12	5	1	15	14	13	4	10	0	7	6	3	9	2	8	11
8	13	11	7	14	12	1	3	9	5	0	15	4	8	6	2	10
9	6	15	14	9	11	3	0	8	12	2	13	7	1	4	10	5
10	10	2	8	4	7	6	1	5	15	11	9	14	3	12	13	0

State Initialization and Padding Scheme in BLAKE-256

The initialization of the state variables (v_i) in the round function of BLAKE-256 is done as:

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ s_0 \oplus k_0 & s_1 \oplus k_1 & s_2 \oplus k_2 & s_3 \oplus k_3 \\ t_0 \oplus k_4 & t_0 \oplus k_5 & t_1 \oplus k_6 & t_1 \oplus k_7 \end{pmatrix}$$

Here, $(h_0, \ldots, h_7)$ are the chain variables, $(k_0, \ldots, k_7)$ are word constants, $(s_0, \ldots, s_3)$ are the 32-bit user-defined salt values and (t_0, t_1) are the counter variables that keep count of bits hashed.

Padding is done to convert any length of provided input/message (l) to a length multiple of 512. The padding scheme is: $m \leftarrow m \| 1000 \ldots 0001 \| \langle l \rangle_{64}$.

Algorithm for BLAKE-256 Hash Function

Algorithm 1: BLAKE-256 Function

```
typedef struct {
    uint32 h[8], s[4], t[2];
    int buflen, nullt;
    uint8 buf[64];
} state256;
Function BLAKE256(string in):
    inlen = length(in);
    uint8 hash[32]; state256 S;
    S.h[0 ... 7] ← IV[0 ... 7];
    uint8 p[n][64] = padding(in, inlen);
    for (i = 0 ... n) do
        ROUND(&S, p[i]);
    end
    hash[0 ... 31] ← S.h[0 ... 7];
End
```

Algorithm 2: Round Function

```
Function ROUND(state256 *S, uint8 *bk):
    uint32 v[16], m[16], i;
    m[0 ... 15] ← bk[0 ... 63];
    Initialize v[0 ... 15];
    for (i = 0 ... 13) do
        G(v, m, i, 0, 4, 8, 12, 0);
        ⋮
    end
    for (i = 0 ... 15) do
        S→h[i % 8] ⊕= v[i];
    end
    for (i = 0 ... 7) do
        S→h[i] ⊕= s[i % 4];
    end
End
```

State Initialization and Padding Scheme in BLAKE2s

The initialization of the state variables ($\mathtt{v_i}$) in the round function of BLAKE2s is done as:

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ IV_0 & IV_1 & IV_2 & IV_3 \\ t_0 \oplus IV_4 & t_1 \oplus IV_5 & f_0 \oplus IV_6 & f_1 \oplus IV_7 \end{pmatrix}$$

Here, $(h_0, \ldots, h_7)$ are the chain variables, (t_0, t_1) are the counter variables that keep count of bytes hashed and (f_0, f_1) are the finalization flags, for the last block of message $f_0 = (ff \ldots ff)$ otherwise $(00 \ldots 00)$ and $f_1 = (00 \ldots 00)$ always.

Padding is done to convert any length of provided input/message (l) to a multiple of 512 bit. The padding scheme is just to append null bytes: $m \leftarrow m \| 000 \ldots 000$.

Parameter Block in BLAKE2s

BLAKE2s uses a parameter block that encodes various parameters. Here, no key, salt and personalization are used, so they are set to null. Values in red are predefined for the sequential mode of BLAKE2s.

Offset	0	1	2	3
0	Digest length	Key length = (00)	Fanout = (01)	Depth = (01)
4	Leaf length = (00000000)			
8	Node Offset = (00000000)			
12	Node Offset(cont.) = (0000)		Node depth = (00)	Inner length = (00)
16 20	Salt = NULL			
24 28	Personalization = NULL			

The parameter block is XORed with the initialization vectors (h_i).

Algorithm for BLAKE2s Hash Function

Algorithm 3: BLAKE2s Function

```
typedef struct {
    uint32 h[8], t[2];
    uint8 buf[64];
    size_t c, outlen;
} ctx;
Function BLAKE2s(string in):
    inlen = length(in);
    uint8 hash[32]; ctx S;
    S.h[0 ... 7] ← IV[0 ... 7];
    uint8 p[n][64] = padding(in, inlen);
    for (i = 0 ... n) do
        ROUND(&S, p[i]);
    end
    hash[0 ... 31] ← S.h[0 ... 7];
End
```

Algorithm 4: Round Function

```
Function ROUND(ctx *S, uint8 *bk, int l):
    uint32 v[16], m[16], i;
    m[0 ... 15] ← bk[0 ... 63];
    Initialize v[0 ... 15];
    if (l ≠ 0) then
        v[14] ← ∼v[14];
    end
    for (i = 0 ... 10) do
        G(v, m, i, 0, 4, 8, 12, 0);
        ⋮
    end
    for (i = 0 ... 7) do
        S→h[i] ⊕= v[i] ⊕ v[i+8];
    end
End
```

The Core Function (G)

The core function in $\mathtt{BLAKE\text{-}256}$ can be represented as: $\mathtt{G}(\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{m}[16],\mathtt{k}[16])$ and each call of core function contains 8 operations as shown below:

$$
\begin{array}{ll}
1{:}\ \hat{a} \leftarrow a + b + (m_{\sigma[r][2i]} \oplus k_{\sigma[r][2i+1]}) & 5{:}\ a' \leftarrow \hat{a} + \hat{b} + (m_{\sigma[r][2i+1]} \oplus k_{\sigma[r][2i]}) \\
2{:}\ \hat{d} \leftarrow (d \oplus \hat{a}) \ggg 16 & 6{:}\ d' \leftarrow (\hat{d} \oplus a') \ggg 8 \\
3{:}\ \hat{c} \leftarrow c + \hat{d} & 7{:}\ c' \leftarrow \hat{c} + d' \\
4{:}\ \hat{b} \leftarrow (b \oplus \hat{c}) \ggg 12 & 8{:}\ b' \leftarrow (\hat{b} \oplus c') \ggg 7
\end{array}
$$

The core function for $\mathtt{BLAKE2s}$ hash function is similar to that of $\mathtt{BLAKE\text{-}256}$ except for the use of constants k_i.

$$
\begin{array}{ll}
1{:}\ \hat{a} \leftarrow a + b + m_{\sigma[r][2i]} & 5{:}\ a' \leftarrow \hat{a} + \hat{b} + m_{\sigma[r][2i+1]} \\
2{:}\ \hat{d} \leftarrow (d \oplus \hat{a}) \ggg 16 & 6{:}\ d' \leftarrow (\hat{d} \oplus a') \ggg 8 \\
3{:}\ \hat{c} \leftarrow c + \hat{d} & 7{:}\ c' \leftarrow \hat{c} + d' \\
4{:}\ \hat{b} \leftarrow (b \oplus \hat{c}) \ggg 12 & 8{:}\ b' \leftarrow (\hat{b} \oplus c') \ggg 7
\end{array}
$$

Visual Representation of the Core Function (G)

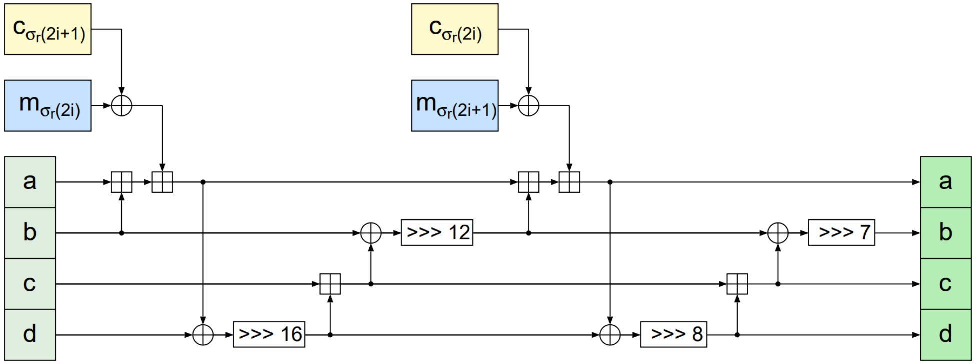

Figure: The core function (G_i). (This image is taken from SHA3 Proposal BLAKE)

Inverting the Core function (G)

The core function ($\mathtt{G}$) takes state variables ($\mathtt{v}_{\mathtt{i}}^{\mathtt{initial}}$) as initial values and modifies them to a final value($\mathtt{v}_{\mathtt{i}}^{\mathtt{final}}$). The equations of a core function can be inverted to derive the initial values of the state variables using the final values as shown below.

$$a = (((d' \lll 8) \oplus a') \lll 16) \oplus d - b - m_i \tag{2}$$

$$a = ((((((b' \lll 7) \oplus c') \lll 12) \oplus b) - c) \lll 16) \oplus d) - m_i - b \tag{3}$$

$$a = a' - ((b' \lll 7) \oplus c') - m_j - b - m_i \tag{4}$$

$$b = (((b' \lll 7) \oplus c') \lll 12) \oplus (c' - d') \tag{5}$$

$$c = c' - d' - ((d' \lll 8) \oplus a') \tag{6}$$

$$c = c' - d' - ((d \oplus (a + b + m_i)) \ggg 16) \tag{7}$$

Inverting the Core Function (G) (cont.)

$$d = (((d' \lll 8) \oplus a') \lll 16) \oplus (a' - ((b' \lll 7) \oplus c') - \mathtt{m}_\mathtt{j}) \tag{8}$$

$$\begin{aligned} a' = &(((((((b' \lll 7) \oplus c') \lll 12) \oplus b) - c) \lll 16) \oplus d) \\ &+ ((b' \lll 7) \oplus c') + \mathtt{m}_\mathtt{j} \end{aligned} \tag{9}$$

$$b' = ((((b \oplus (c' - d')) \ggg 12) \oplus c') \ggg 16) \tag{10}$$

$$d' = c' - c - ((d \oplus (a + b + \mathtt{m}_\mathtt{i})) \ggg 16) \tag{11}$$

It can be observed that the values of $\mathtt{m}_\mathtt{i}$ and $\mathtt{m}_\mathtt{j}$ can be derived using eq.(2), eq.(3) and eq.(9) from the values of (a, b, c, d) and (a', b', c', d').

Preimage Attack on 1.5 Rounds of BLAKE-256 (or BLAKE2s)

Given the states v_i^0 and $v_i^{1.5}$ (where v_i^r represent the values of state variables after r rounds of the hash function), the messages can be derived using the following steps:

Guess m_8, m_{10}, m_{11} and $v_{10}^{0.5}$ and determine variables using equations as shown below.

	Determine	Eq.	Determine	Eq.	Determine	Eq.	Determine	Eq.
1.	$v_4^1, \ldots, v_7^1$	(5)	$v_8^1, \ldots, v_{11}^1$	(6)	v_{12}^1, v_{13}^1	(8)	$v_6^{0.5}, v_7^{0.5}$	(5)
2.	m_4	(3)	v_1^1	(3)	$v_{14}^{0.5}$	(7)	$v_1^{0.5}$	(4)
3.	$v_{11}^{0.5}$	(6)	$v_{12}^{0.5}$	(3)	$v_2^{0.5}$	(6)	m_5	(9)
4.	m_6	(3)	v_{15}^1	(8)	$v_{15}^{0.5}$	(7)	$v_5^{0.5}$	(5)
5.	v_0^1	(6)	m_9	(9)	m_{14}	(3)	$v_3^{0.5}$	(6)
6.	m_7	(9)	$v_0^{0.5}$	(3)	$v_8^{0.5}$	(6)	m_0	(2)
7.	v_2^1	(6)	v_{14}^1	(3)	m_{15}	(9)	$v_4^{0.5}$	(10)
8.	m_1	(9)	$v_9^{0.5}$	(7)	v_3^1	(9)	m_{13}	(3)
9.	m_2	(3)	m_3	(9)	$v_{13}^{0.5}$	(8)	m_{12}	(3)

If $f_{v_i^0}^{1.5}(m) = v_i^{1.5}$, output m, otherwise make a new guess.

Analysis of the Inverted Round Preimage Attack

Using the above method, the preimage (or message) of any given hash value for 1.5-round `BLAKE-256` or `BLAKE2s` can be derived with time complexity of 2^{128} due to random guessing of 128 (4×32) bits.

Limitations of the method:

- In this method, at least 4 words need to be guessed (or predetermined).
- In this attack, increasing the rounds of the hash function by 0.5 increases the number of guesses required exponentially, which results in an exponential increase in time complexity.
- For e.g., for 1.5 rounds, no. of guesses required is $4(2^2)$, but for 2 rounds, no. of guesses required is $8(2^3)$.

Preimage Attack on 2 rounds of BLAKE-256 (or BLAKE2s)

Here, the data given are the initial value h_i^{t-1} and the final hash value $h^t = h_0^t, \dots, h_7^t$. We need to find the message (m) used.

It can be observed that when modifying the initial value $h_i^{t-1} (i = 0, \dots, 7)$, the state words of v after the G function can be kept unaffected by modifying the message words $m_{\sigma[0][2i]}$.

Init	$(\hat{v}_0, v_4, v_8, v_{12})$ $\hat{h}_0^{t-1}$	$(v_1, \hat{v}_5, v_9, v_{13})$ $\hat{h}_5^{t-1}$	$(\hat{v}_2, v_6, v_{10}, v_{14})$ $\hat{h}_2^{t-1}$	$(v_3, v_7, v_{11}, v_{15})$
R 0.5	$G_0(v_0, v_4, v_8, v_{12})$ $\hat{m}_0$	$G_1(v_1, v_5, v_9, v_{13})$ $\hat{m}_2$	$G_2(v_2, v_6, v_{10}, v_{14})$ $\hat{m}_4$	$G_3(v_3, v_7, v_{11}, v_{15})$
R 1	$G_4(v_0, v_5, v_{10}, v_{15})$	$G_5(v_1, v_6, v_{11}, v_{12})$	$G_6(v_2, v_7, v_8, v_{13})$	$G_7(v_3, v_4, v_9, v_{14})$
R 1.5	$G_0(v_0, v_4, v_8, v_{12})$	$G_1(\hat{v}_1, \hat{v}_5, \hat{v}_9, \hat{v}_{13})$ $\hat{m}_4$	$G_2(v_2, v_6, v_{10}, v_{14})$	$G_3(v_3, v_7, v_{11}, v_{15})$
R 2	$G_4(v_0, v_5, v_{10}, v_{15})$	$G_5(\hat{v}_1, \hat{v}_6, \hat{v}_{11}, \hat{v}_{12})$ $\hat{m}_0, \hat{m}_2$	$G_6(v_2, v_7, v_8, v_{13})$	$G_7(v_3, v_4, v_9, v_{14})$

Theory of Attack

The final hash values are calculated as follows(neglecting salt values):

$$
\begin{array}{ll}
h'_0 \leftarrow h_0 \oplus v_0 \oplus v_8 & h'_4 \leftarrow h_4 \oplus v_4 \oplus v_{12} \\
h'_1 \leftarrow h_1 \oplus v_1 \oplus v_9 & h'_5 \leftarrow h_5 \oplus v_5 \oplus v_{13} \\
h'_2 \leftarrow h_2 \oplus v_2 \oplus v_{10} & h'_6 \leftarrow h_6 \oplus v_6 \oplus v_{14} \\
h'_3 \leftarrow h_3 \oplus v_3 \oplus v_{11} & h'_7 \leftarrow h_7 \oplus v_7 \oplus v_{15}
\end{array}
$$

As the values of v_0 and v_8 are kept unaffected by modifying m_0, according to the finalization: $h'_0 \leftarrow \hat{h}_0^{t-1} \oplus v_0 \oplus v_8$, the output h'_0 is changed by the value of $\hat{h}_0^{t-1}$ directly or we can control the word of hash value h'_0 after 2 rounds.

Similarly, after 2 rounds, the values of v_5 and v_{13} are unaffected due to modified m_2. Thus, from $h'_5 \leftarrow \hat{h}_5^{t-1} \oplus v_5 \oplus v_{13}$ we get that the value of h'_5 can be controlled with the value of $\hat{h}_5^{t-1}$ alone.

Theory of Attack (Cont.)

Again, after 2 rounds, we can observe from $h'_2 \leftarrow \hat{h}_2^{t-1} \oplus v_2 \oplus v_{10}$ that the value of h'_2 can be controlled with the value of $\hat{h}_2^{t-1}$ alone as the values of v_2 and v_{10} are unaffected due to modified m_4.

The steps to be followed in the attack are:

- Set message words $m = m_0, \ldots, m_{15}$ randomly.
- Calculate the hash value of 2 rounds: $h' = \mathtt{compress}_{2R}(h^{t-1}, m, s, t)$.
- Set the value of $\hat{h}_n^{t-1}(n = 0, 2(\text{or } 5))$ according to $\hat{h}_n^{t-1} = h_n^{t-1} \oplus h'_n \oplus h_n^t$.
- Calculate the value of m_0, m_4 (or m_2) by using eq. (12), (13) (or (14)) respectively.
- Modify m_j to $\hat{m_j}(j = 0, 4(\text{or } 2))$ and calculate $h' = \mathtt{compress}_{2R}(\hat{h}^{t-1}, \hat{m}, s, t)$, then we can get $h'_n = h_n^t(n = 0, 2(\text{or } 5))$ again.
- The steps are repeated until the left 6 words' hash value equates to the given h^t.

Equations to Modify Messages

The first operation of the core function ($G_{0.5}$) is reversed to calculate the value of $m_i (i = 0, 4(\text{or } 2))$ every time the initial values or $\hat{h}_n^{t-1} (n = 0, 2(\text{or } 5))$ are changed as follows:

$$h_0^{t-1} + h_4^{t-1} + (m_0 \oplus c_1) = \hat{h}_0^{t-1} + h_4^{t-1} + (\hat{m_0} \oplus c_1)$$

$$\hat{m_0} = (h_0^{t-1} + (m_0 \oplus c_1) - \hat{h}_0^{t-1}) \oplus c_1 \tag{12}$$

$$h_2^{t-1} + h_6^{t-1} + (m_4 \oplus c_5) = \hat{h}_2^{t-1} + h_6^{t-1} + (\hat{m_4} \oplus c_5)$$

$$\hat{m_4} = (h_2^{t-1} + (m_4 \oplus c_5) - \hat{h}_4^{t-1}) \oplus c_5 \tag{13}$$

$$h_1^{t-1} + h_5^{t-1} + (m_2 \oplus c_3) = h_1^{t-1} + \hat{h}_5^{t-1} + (\hat{m_2} \oplus c_3)$$

$$\hat{m_2} = (h_5^{t-1} + (m_2 \oplus c_3) - \hat{h}_5^{t-1}) \oplus c_3 \tag{14}$$

Preimage Attack on 2.5 rounds of BLAKE-256 (or BLAKE2s)

Mounting the same attack on 2.5 rounds of BLAKE, it can be observed that modifying any of the initial values $h_i^{t-1}(i = 0, \ldots, 7)$ and its corresponding message (m), changes state variables (v) required for hash calculation in later rounds.

Init	$(\hat{v}_0, v_4, v_8, v_{12})$ $\hat{h}_0^{t-1}$	$(\hat{v}_1, v_5, v_9, v_{13})$ $\hat{h}_1^{t-1}$	$(\hat{v}_2, v_6, v_{10}, v_{14})$	$(v_3, v_7, v_{11}, v_{15})$
R 0.5	$G_0(v_0, v_4, v_8, v_{12})$ $\hat{m}_0$	$G_1(v_1, v_5, v_9, v_{13})$ $\hat{m}_2$	$G_2(v_2, v_6, v_{10}, v_{14})$	$G_3(v_3, v_7, v_{11}, v_{15})$
R 1	$G_4(v_0, v_5, v_{10}, v_{15})$	$G_5(v_1, v_6, v_{11}, v_{12})$	$G_6(v_2, v_7, v_8, v_{13})$	$G_7(v_3, v_4, v_9, v_{14})$
R 1.5	$G_0(v_0, v_4, v_8, v_{12})$	$G_1(v_1, v_5, v_9, v_{13})$	$G_2(v_2, v_6, v_{10}, v_{14})$	$G_3(v_3, v_7, v_{11}, v_{15})$
R 2	$G_4(v_0, v_5, v_{10}, v_{15})$	$G_5(\hat{v}_1, \hat{v}_6, \hat{v}_{11}, v_{12})$ $\hat{m}_0, \hat{m}_2$	$G_6(v_2, v_7, v_8, v_{13})$	$G_7(v_3, v_4, v_9, v_{14})$
R 2.5	$G_0(v_0, v_4, v_8, v_{12})$	$G_1(\hat{v}_1, \hat{v}_5, \hat{v}_9, \hat{v}_{13})$ $\hat{m}_0$	$G_1(\hat{v}_2, \hat{v}_6, \hat{v}_{10}, \hat{v}_{14})$ $\hat{m}_2$	$G_3(v_3, v_7, v_{11}, v_{15})$

Analysis of the Attack - Result

- The modified messages ($\mathtt{m}_0, \mathtt{m}_4(\mathtt{or}\ \mathtt{m}_2)$) modify such sets of state words ($\mathtt{v}$) in the later rounds that are not used for final hash value calculation for either $\mathtt{h}'_0, \mathtt{h}'_2$ (or $\mathtt{h}'_5$).
- Under this attack, we need to follow the steps until the left 6 words(6×32 bits) of the generated hash value ($\mathtt{h}'$) match with the given hash value ($\mathtt{h}^t$).
- The time complexity of the attack is: $2^{6\times32} = 2^{192}$, which is an improvement by 2^{32} over previously-available attack (complexity $= 2^{224}$).
- Extending the attack to 2.5 rounds, the complexity increases to 2^{224} and remains 2^{224} until 3.5 rounds.
- The complexity increases to 2^{256} for rounds more than 3.5 and then it becomes a brute-force attack.

Work to be done: The above method is verified theoretically only. A new verification method can be devised that can verify the attack practically.

References

[1] Ronald L. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, April 1992. https://www.ietf.org/rfc/rfc1321.txt

[2] Jean-Philippe Aumasson, Luca Henzen, Willi Meier and Raphael C.-W. Phan. SHA-3 proposal BLAKE version 1.3, December 16, 2010. https://www.aumasson.jp/blake/blake.pdf.

[3] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn and Christian Winnerlein. BLAKE2: simpler, smaller, fast as MD5. Cryptology ePrint Archive, Paper 2013/322, 2013. https://eprint.iacr.org/2013/322.

[4] Bernstein, D.J. ChaCha, a variant of Salsa20. http://cr.yp.to/chacha.html.

[5] LI Ji and XU Liangyu. Attacks on round-reduced blake. Cryptology ePrint Archive, Paper 2009/238, 2009. https://eprint.iacr.org/2009/238.

[6] Jean-Philippe Aumasson, Jian Guo, Simon Knellwolf, Krystian Matusiewicz, and Willi Meier. Differential and invertibility properties of blake (full version). Cryptology ePrint Archive, Paper 2010/043, 2010. https://eprint.iacr.org/2010/043.