# The Dynamic SHA2 Hash Function

Xu ZiJie
xuzijiewz@gmail.com

## Outline

1. Introduction
2. Design considerations / Responses
3. Dynamic SHA2
4. Implementations
5. Security Analysis
6. Improvements to resist known attack

# 1. Introduction

## Why design Dynamic SHA2?

Biham and Shamir discovere Differential cryptanalysis (1990). Considers step-by-step ``difference’’ (XOR) between two computations…

Wang used Differential cryptanalysis to break MD5；

## NIST SHA-3 competition

Input: 0 to $2^{64}$-1 bits, size not known in advance

Output sizes 224, 256, 384, 512 bits

Preimage resistance.

Second preimage resistance.

Collision-resistance.

Pseudorandomness, simplicity, flexibility, speedy, …

## 2. Design Considerations / Responses

# How to resist Differential analysis?

If the difference of working variables between two computations is very complex, it will be hard to analyse difference. There are two ways that we can use to resist differential analysis :

More rounds or steps. After more rounds, the difference will be more complex. It is harder to control the difference of working variables.

Use the functions that need huge ANFs to describe. And it need huge ANFs to describe Data-Depend functions. So Dynamic SHA use Data-Depend functions to resist differential analysis.

# What is new in Dynamic SHA2?

Dynamic SHA2 has data-depend function R, G and data-depend rotate operation. When input different message, the different calculation will be done. The (one block) message value space is divided into $2^{256}$ (resp. $2^{512}$) parts. There is one message value in a part.

The ANFs that describe function R, G, data-depend rotate operation and function R1 has up to $2^{261}$(resp. $2^{520}$),9,243 (resp.729), $2^{229}$(resp. $2^{454}$) items. This will make the difference of working variables very complex.

# 3. Dynamic SHA2

## 3.2.2. Operations

Dynamic SHA2-224/256 operations on 32-bit words.

Dynamic SHA2-384/512 operations on 64-bit words.

The following operations:

| | | |
|---|---|---|
| $+$ | | modulo $2^{32}$ or modulo $2^{64}$ . |
| $\wedge$ | | AND. |
| $\vee$ | | OR. |
| $\oplus$ | | XOR. |
| $\neg$ | | Negation. |
| $SHR^n(x)$ | x>>n | shift right operation. |
| $SHL^n(x)$ | x<<n | shift left operation. |
| $ROTR^n(x)$ | x>>>n | rotate left (circular left shift) operation. |

# 3.1 Functions and Constants

## 3.1.1. Initial Hash Value

| Dynamic SHA2-224 | Dynamic SHA2-256 |
|---|---|
| $H_0^{(0)}$=0xc1059ed8,<br>$H_1^{(0)}$=0x367cd507,<br>$H_2^{(0)}$=0x3070dd17,<br>$H_3^{(0)}$=0xf70e5939,<br>$H_4^{(0)}$=0xffc00b31,<br>$H_5^{(0)}$=0x68581511,<br>$H_6^{(0)}$=0x64f98fa7,<br>$H_7^{(0)}$=0xbefa4fa4, | $H_0^{(0)}$=0x6a09e667,<br>$H_1^{(0)}$=0xbb67ae85,<br>$H_2^{(0)}$=0x3c6ef372,<br>$H_3^{(0)}$=0xa54ff53a,<br>$H_4^{(0)}$=0x510e527f,<br>$H_5^{(0)}$=0x9b05688c,<br>$H_6^{(0)}$=0x1f83d9ab,<br>$H_7^{(0)}$=0x5be0cd19, |

**Table 1. Initial Hash Value of Dynamic SHA2-224/256**

| Dynamic SHA2-384 | Dynamic SHA2-512 |
|---|---|
| $H_0^{(0)}$=0xcbbb9d5dc1059ed8,<br>$H_1^{(0)}$=0x629a292a367cd507,<br>$H_2^{(0)}$=0x9159015a3070dd17,<br>$H_3^{(0)}$=0x152fecd8f70e5939,<br>$H_4^{(0)}$=0x67332667ffc00b31,<br>$H_5^{(0)}$=0x8eb44a8768581511,<br>$H_6^{(0)}$=0xdb0c2e0d64f98fa7,<br>$H_7^{(0)}$=0x47b5481dbefa4fa4, | $H_0^{(0)}$=0x6a09e667f3bcc908,<br>$H_1^{(0)}$=0xbb67ae8584caa73b,<br>$H_2^{(0)}$=0x3c6ef372fe94f82b,<br>$H_3^{(0)}$=0xa54ff53a5f1d36f1,<br>$H_4^{(0)}$=0x510e527fade682d1,<br>$H_5^{(0)}$=0x9b05688c2b3e6c1f,<br>$H_6^{(0)}$=0x1f83d9abfb41bd6b,<br>$H_7^{(0)}$=0x5be0cd19137e2179, |

**Table 2. Initial Hash Value of Dynamic SHA2-384/512**

## 3.1.2. Function G

Logical function G operates on three words a,b,c and a 2-bit word t, and produces a word y as output.

$$y=\begin{cases} G_{t=00}(a,b,c)=a\oplus b\oplus c \\ G_{t=01}(a,b,c)=(a\wedge b)\oplus c \\ G_{t=10}(a,b,c)=(\neg(a\vee c))\vee(a\wedge(b\oplus c)) \\ G_{t=11}(a,b,c)=(a\vee\neg c)\vee(\neg(a\vee(b\oplus c))) \end{cases}$$

It can describe function G with Algebraic Normal Form(ANF) and Numerical Normal Form (NNF) as follow, let $t=(t_1,t_0)$:

$$y_i=\mathrm{a}_i\oplus\mathrm{b}_i\oplus\mathrm{c}_i\oplus\mathrm{t}_1\oplus\mathrm{a}_i\mathrm{t}_0\oplus\mathrm{b}_i\mathrm{t}_1\oplus\mathrm{a}_i\mathrm{b}_i\mathrm{t}_0\oplus\mathrm{a}_i\mathrm{b}_i\mathrm{t}_1\oplus\mathrm{a}_i\mathrm{c}_i\mathrm{t}_1\mathrm{t}_0$$

$$\begin{aligned} y_i = {} & \mathrm{a}_i+\mathrm{b}_i+\mathrm{c}_i+\mathrm{t}_1-2\mathrm{a}_i\mathrm{b}_i-2\mathrm{a}_i\mathrm{c}_i-2\mathrm{b}_i\mathrm{c}_i-\mathrm{a}_i\mathrm{t}_0-2\mathrm{a}_i\mathrm{t}_1-2\mathrm{b}_i\mathrm{t}_0 \\ & -\mathrm{b}_i\mathrm{t}_1-2\mathrm{c}_i\mathrm{t}_1+4\mathrm{a}_i\mathrm{b}_i\mathrm{c}_i+3\mathrm{a}_i\mathrm{b}_i\mathrm{t}_0+3\mathrm{a}_i\mathrm{b}_i\mathrm{t}_1+2\mathrm{a}_i\mathrm{c}_i\mathrm{t}_0 \\ & +4\mathrm{a}_i\mathrm{c}_i\mathrm{t}_1+2\mathrm{b}_i\mathrm{c}_i\mathrm{t}_1+2\mathrm{a}_i\mathrm{t}_0\mathrm{t}_1-6\mathrm{a}_i\mathrm{b}_i\mathrm{c}_i\mathrm{t}_0-6\mathrm{a}_i\mathrm{b}_i\mathrm{c}_i\mathrm{t}_1-4\mathrm{a}_i\mathrm{b}_i\mathrm{t}_0\mathrm{t}_1 \\ & -3\mathrm{a}_i\mathrm{c}_i\mathrm{t}_1\mathrm{t}_0+6\mathrm{a}_i\mathrm{b}_i\mathrm{c}_i\mathrm{t}_0\mathrm{t}_1 \end{aligned}$$

$y_i,a_i,b_i,c_i$ is i-th bit of y,a,b,c.

- Ps: Claude Carlet had recounted NNF in “Boolean Functions for Cryptography and Error Correcting Codes”[2008]. NNF and ANF is equivalent, NNF and ANF can be transformed each other.

## 3.1.3. Function R1

Function R1 operates on eight words a,b,c,d,e,f,g,h, produces a word y as output.

| | |
|---|---|
| Dynamic SHA2-224/256 | $t=(((((a+b)\oplus c)+d)\oplus e)+f)\oplus g$<br>$t=(SHR^{17}(t)\oplus t)\wedge(2^{17}-1)$<br>$t=(SHR^{10}(t)\oplus t)\wedge(2^{10}-1)$<br>$t=(SHR^{5}(t)\oplus t)\wedge 31$<br>$y=ROTR^{t}(h)$ |
| Dynamic SHA2-384/512 | $t=(((((a+b)\oplus c)+d)\oplus e)+f)\oplus g$<br>$t=(SHR^{36}(t)\oplus t)\wedge(2^{36}-1)$<br>$t=(SHR^{18}(t)\oplus t)\wedge(2^{18}-1)$<br>$t=(SHR^{12}(t)\oplus t)\wedge(2^{12}-1)$<br>$t=(SHR^{6}(t)\oplus t)\wedge 63$<br>$y=ROTR^{t}(h)$ |

The ANFs that describe function R have up to $2^{229}$(resp. $2^{454}$) items. This will make the difference of output very complex.

The profit/cost of function R1 is very high. 

### 3.1.4. Function R

Function R operates on eight words a,b,c,d,e,f,g,h and a 5-bit(resp 6-bit) word t, produces a word y as output.

$$y= ROTR^{t}(((((((a \oplus b)+c) \oplus d)+e) \oplus f)+g) \oplus h)$$

The ANFs that describe function R have up to $2^{261}$(resp. $2^{520}$) items. This will make the difference of output very complex.

The profit/cost of function R is very high. 

### 3.1.5. Function COMP

Function COMP operates on eight working variables a,b,c,d,e,f,g,h, eight message word $x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7$ and an integer t.

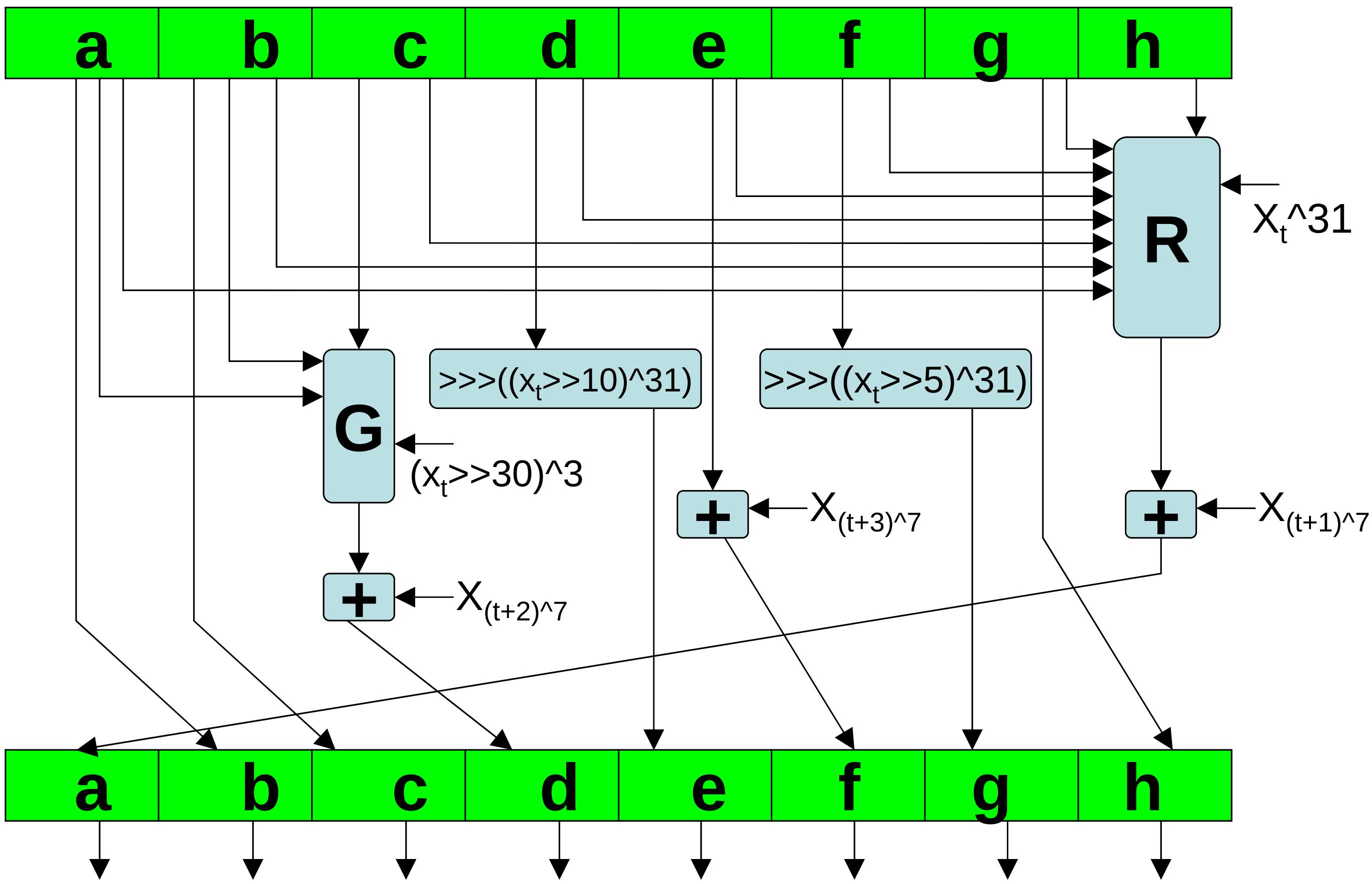


**Fig 1.a. Function COMP for Dynamic SHA2-224/256**

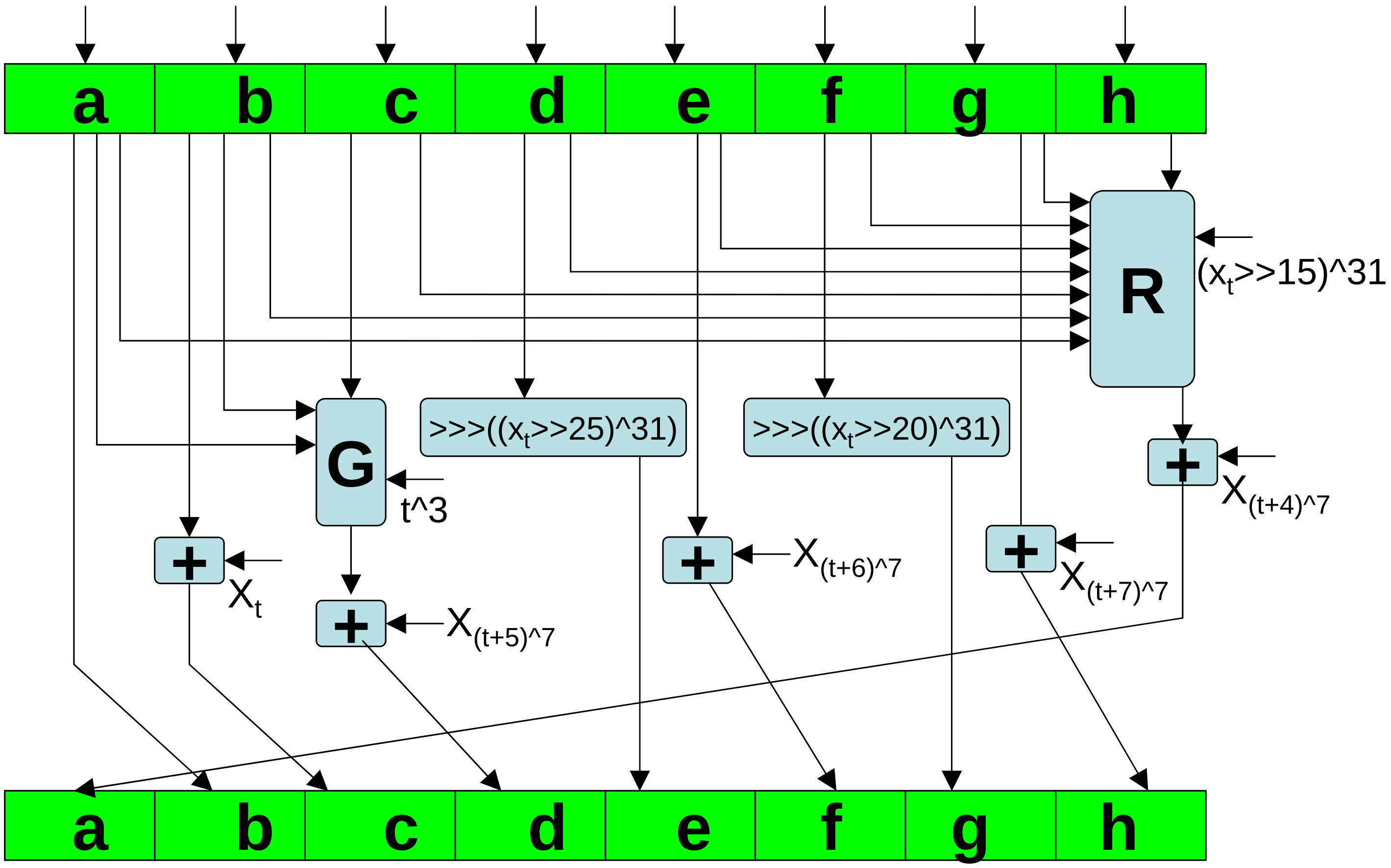


**Fig 1.b. Function COMP for Dynamic SHA2-224/256**

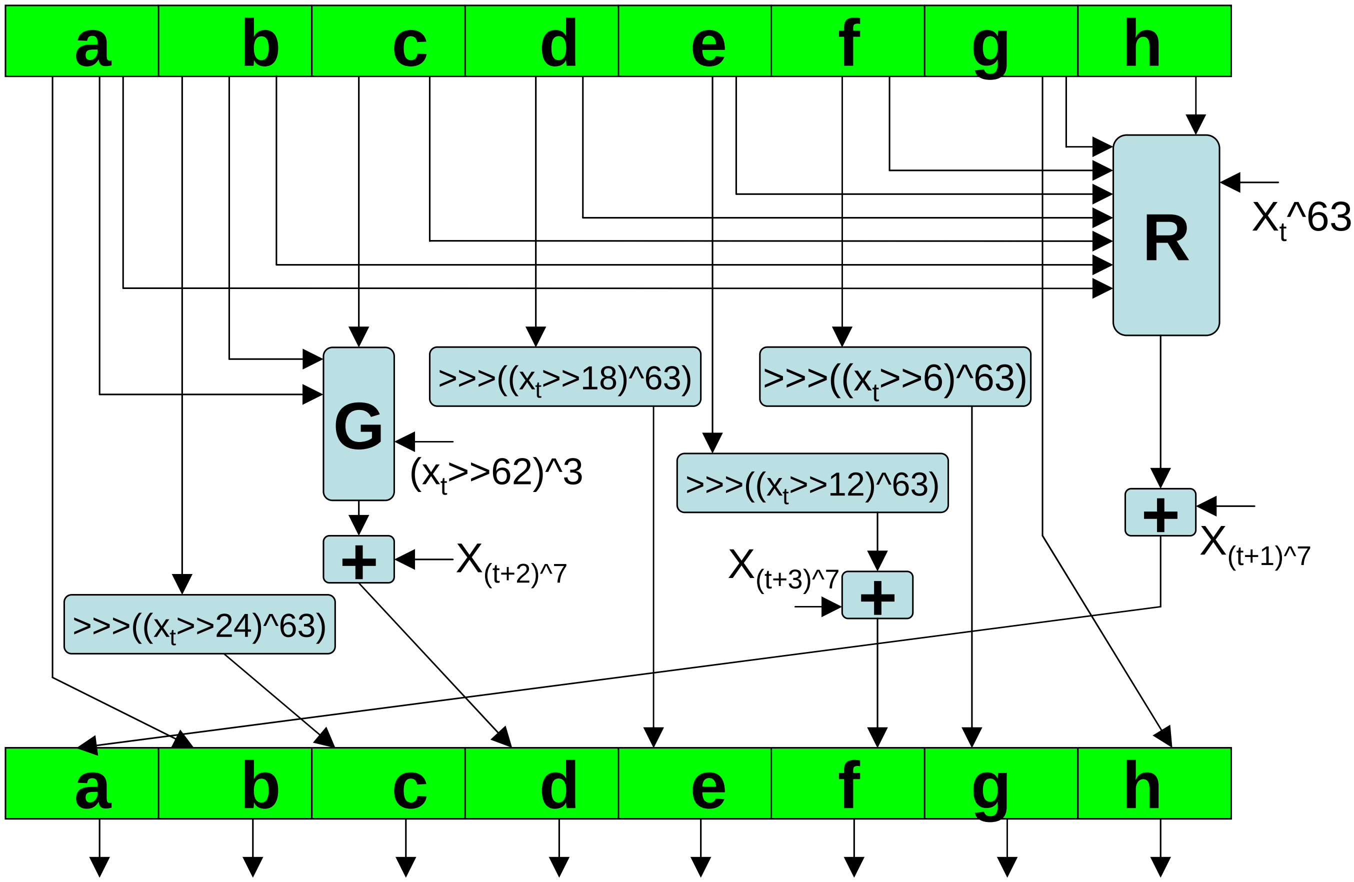


**Fig 2.a. Function COMP for Dynamic SHA2-384/512**

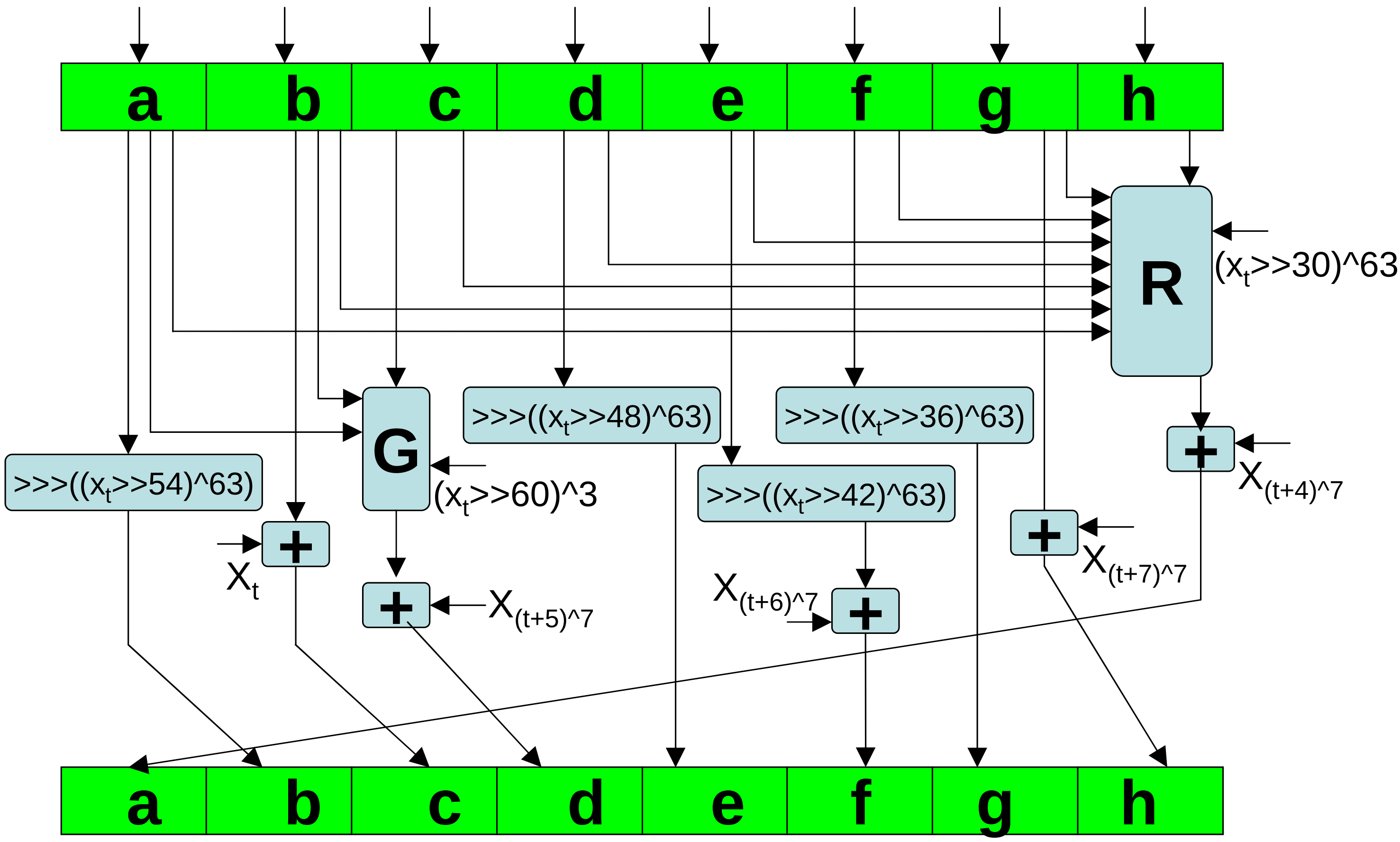


**Fig 2.b. Function COMP for Dynamic SHA2-384/512**

# 3.2 Preprocessing

At first, the message will be paded and divided into some blocks. These blocks will be inputed compress function $H_{DSHA2}$ in order. The output of the last compution will be truncated as 224/256/384/512 bits.

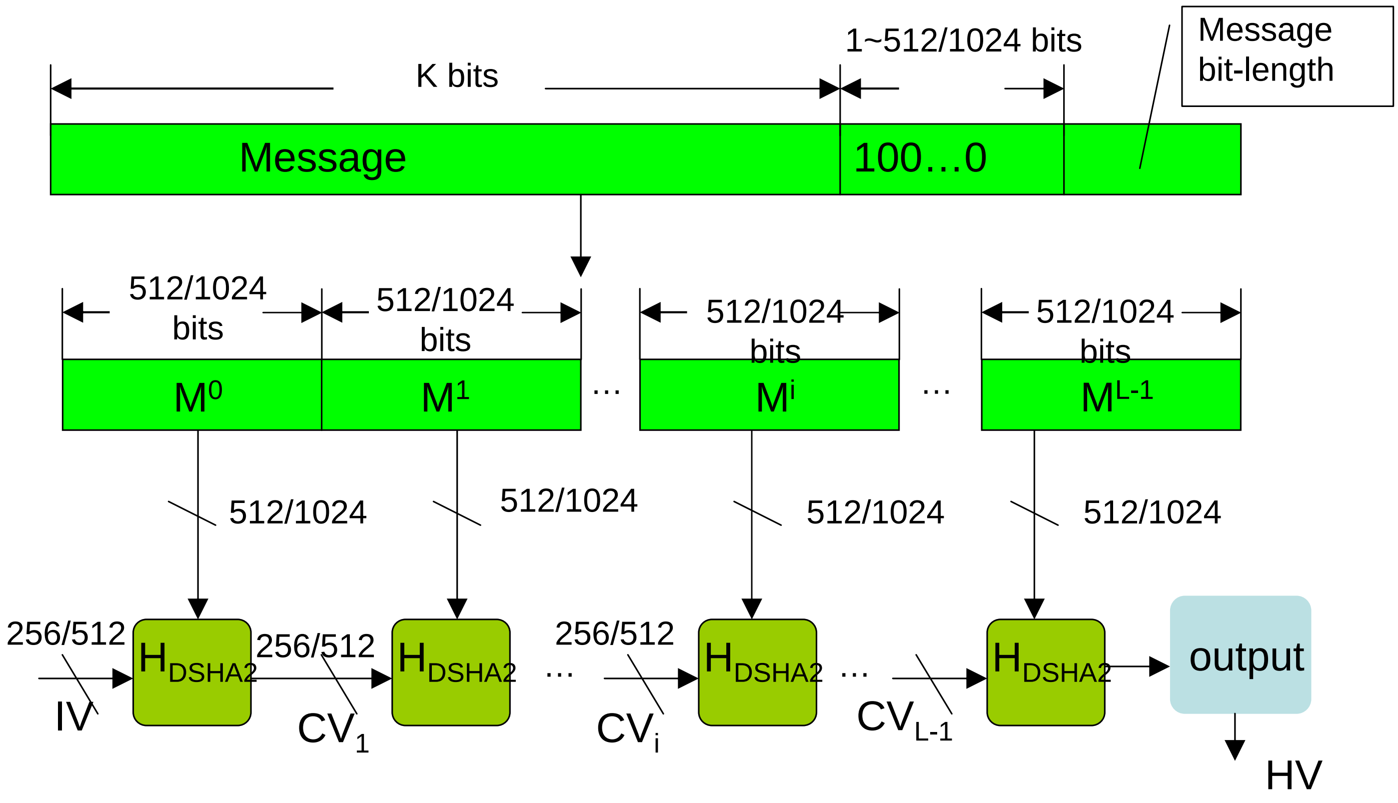


**Fig 3. Preprocessing of Dynamic SHA2**

# 3.2.1 Compression function

## Compression function inputs

16 words (512/1024 bits) data block

8 words (256/512 bits) chaining values

## Compression function outputs

8 words (256/512 bits) chaining values

Compression function include three iterative parts.

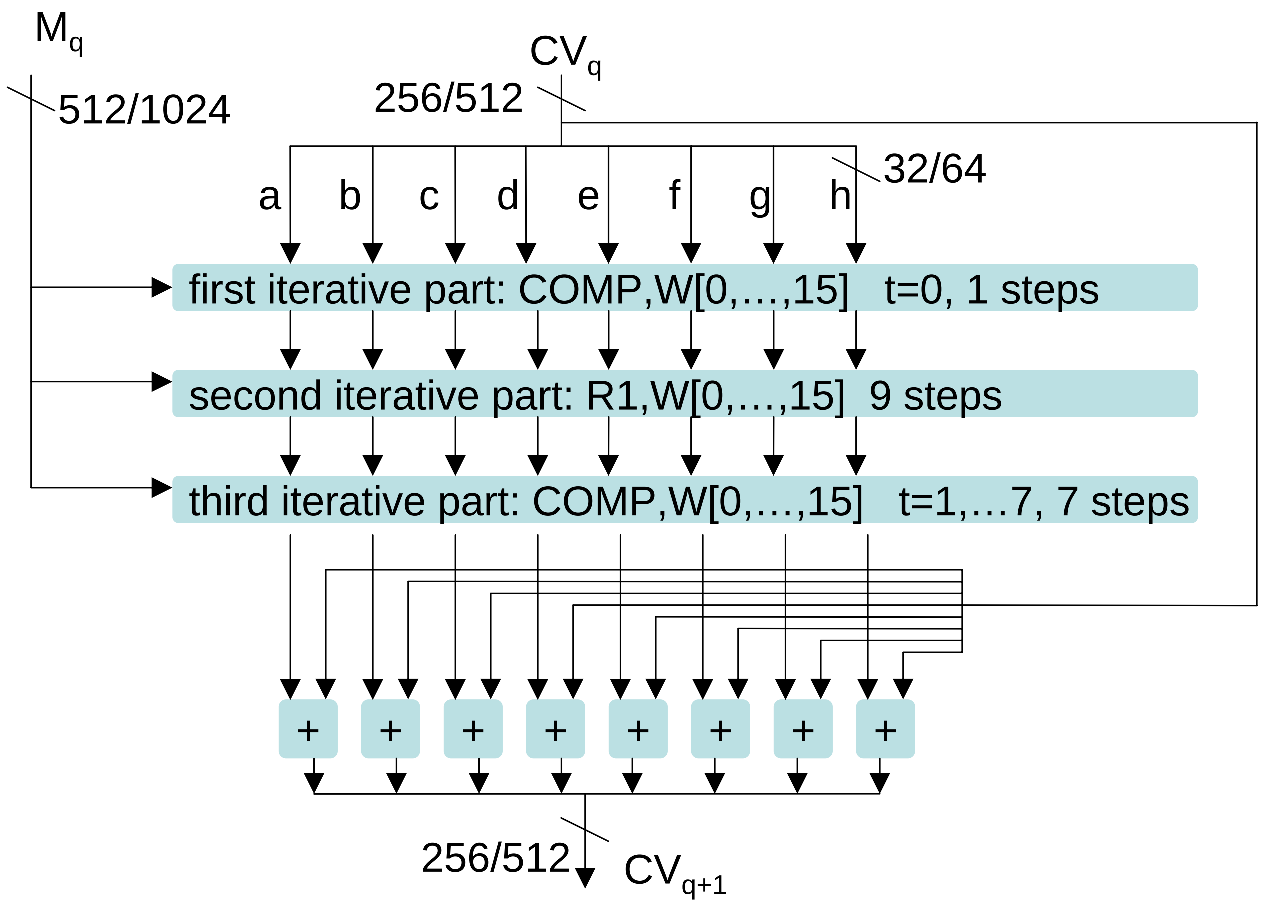


**Fig 4. Compression function of Dynamic SHA2**

## One step compute of first/third iterative part

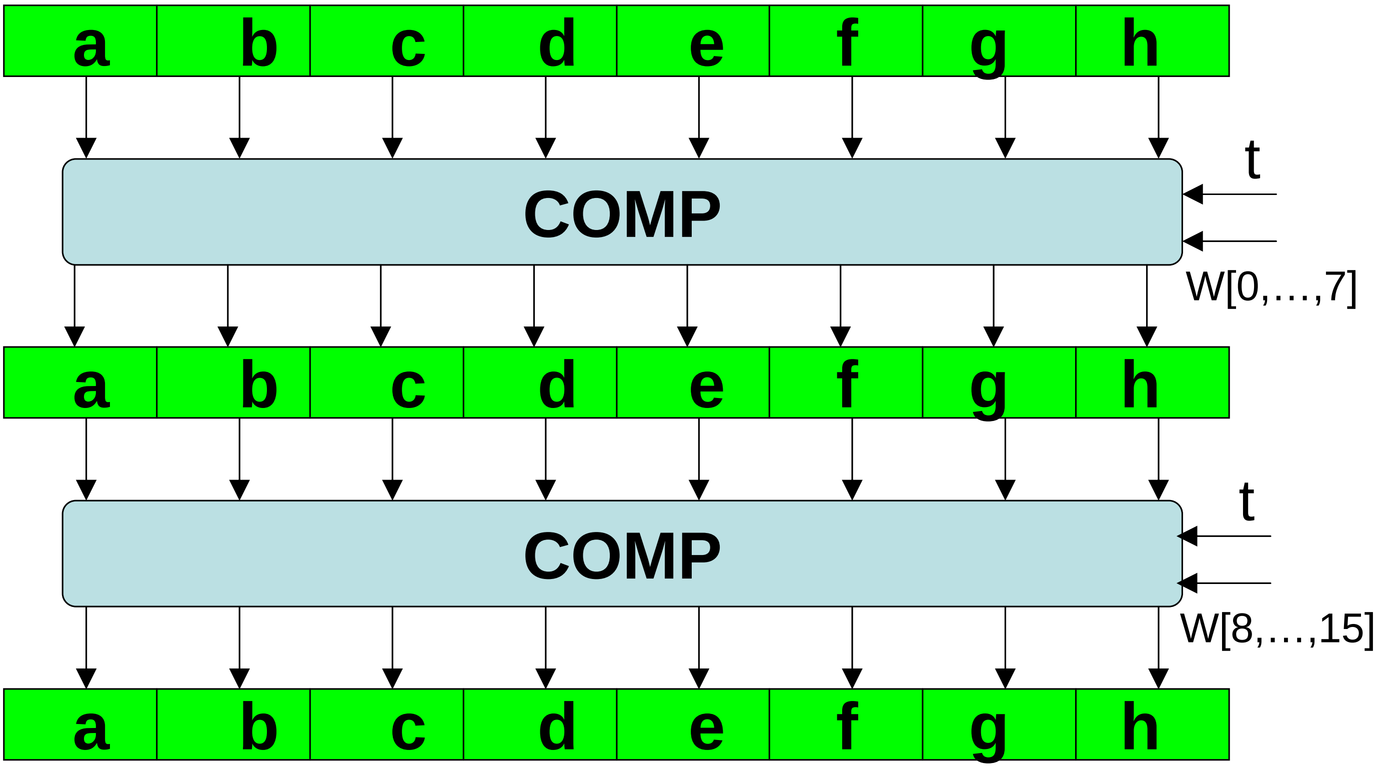


**Fig 5. One step compute of first/third iterative parts of of Dynamic SHA2**

## One step compute of second iterative part

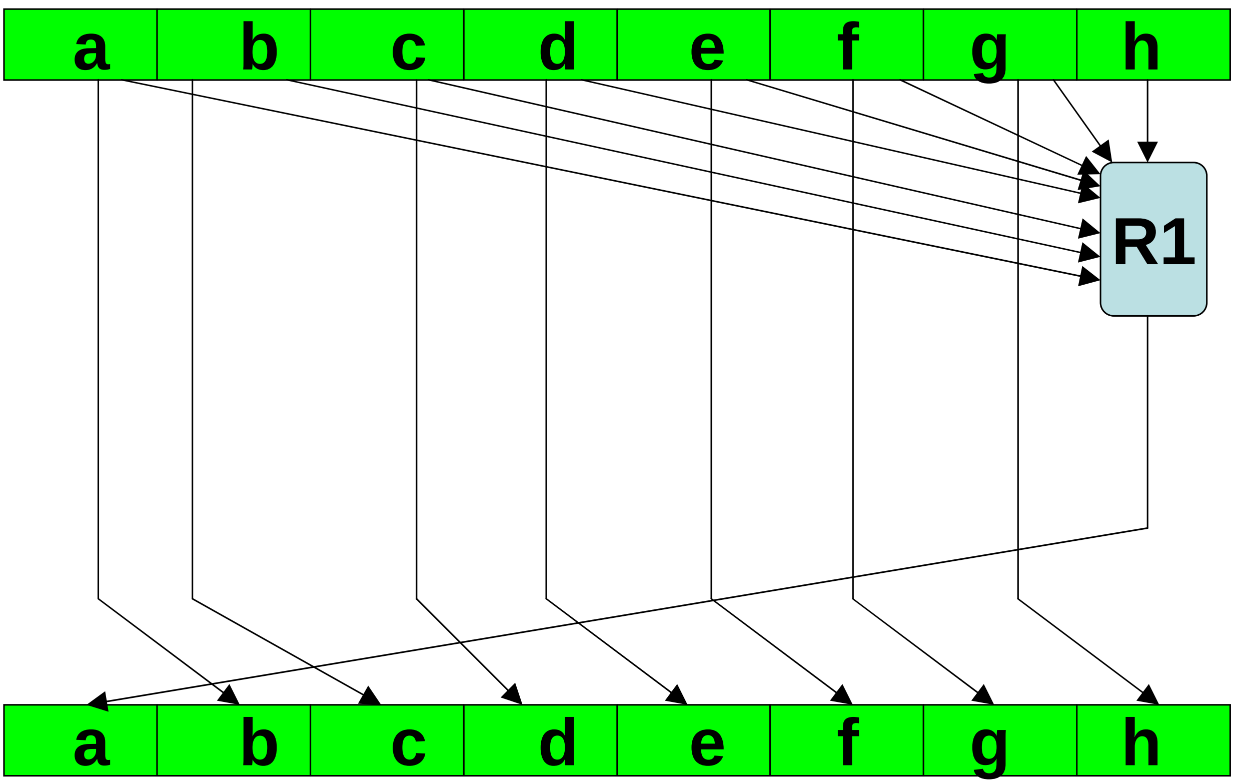


**Fig 6. One step compute of second iterative parts of of Dynamic SHA2**

# Compression function code:

**Input:** $H_0^i, H_1^i, H_2^i, H_3^i, H_4^i, H_5^i, H_6^i, H_7^i, w_0, \ldots, w_{15}$

$a=H_0^i$; $b=H_1^i$; $c=H_2^i$; $d=H_3^i$; $e=H_4^i$; $f=H_5^i$; $g=H_6^i$; $h=H_7^i$;

COMP(a,b,c,d,e,f,g,h,$w_0, \ldots, w_7$,0);

COMP(a,b,c,d,e,f,g,h,$w_8, \ldots, w_{15}$,0);

**for** i =0 **to** **8**

{

T=R1(a,b,c,d,e,f,g,h);

h=g; g=f; f=e; e=d; c=b; b=a; a=T;

}

**for** **t**=*1* **to** **7**

{

COMP(a,b,c,d,e,f,g,h,$w_0, \ldots, w_7$,t);

COMP(a,b,c,d,e,f,g,h,$w_8, \ldots, w_{15}$,t);

}

**return** $H_0^i = H_0^i + a$; $H_1^i = H_1^i + b$; $H_2^i = H_2^i + c$; $H_3^i = H_3^i + d$;

$H_4^i = H_4^i + e$; $H_5^i = H_5^i + f$; $H_6^i = H_6^i + g$; $H_7^i = H_7^i + h$;

# 4. Implementations

Dynamic SHA  had been implemented on follow platform:

•Xilinx

•Keil uVision, the processor is Intel 80/87c58

•Wintel personal computer, with an Intel Core 2 Duo Processor, 2.4GHz clock speed, 2GB RAM, running Windows Vista Ultimate 32-bit (x86) Edition.

The compiler is: The ANSI C compiler in the Microsoft Visual Studio 2005 Professional Edition.

•Wintel personal computer, with an Intel Core 2 Duo Processor, 2.4GHz clock speed, 2GB RAM, running Windows Vista Ultimate 64-bit (x64) Edition.

The compiler is: The ANSI C compiler in the Microsoft Visual Studio 2005 Professional Edition.

# 5. Security Analysis

# Birthday attack resistance

Birthday attack resistance of Dynamic SHA2 as follow:

| | |
|---|---|
| Dynamic SHA2-224 | $2^{-112}$ |
| Dynamic SHA2-256 | $2^{-128}$ |
| Dynamic SHA2-384 | $2^{-192}$ |
| Dynamic SHA2-512 | $2^{-256}$ |

## When Dynamic SHA2 is attacked by differential attacks, what will happen?

The ANFs that describe function R1,R,G,data-depend rotate operation have up to $2^{229}$ (resp. $2^{454}$),$2^{261}$ (resp. $2^{520}$), 9, 243 (resp.729) items. The difference of working variables will be very complex.

If attacker guess the parameter in function R,G, data-depend rotate operation to reduce complexity, this will divide the message space to $2^{512}$ (resp. $2^{1024}$) parts. There is one message value in a part. In different part, the calculation is different. It can not find the collision in a part.

## When Dynamic SHA2 is attacked by extension attack and multicollision attack, what will happen?

In extension attack and multicollision attack, it need find collision of hash algorithm. The probability of find collision of Dynamic SHA2 is $2^{-n/2}$, where n is output sizes in bits.

The hash value of Dynamic SHA2-224/384 is truncated. In keyed hash, , it can not get all bits of last compute. It is hard to be attacked by extension attack.

There are some ways that resist length extension attack and multicollision attack. HMAC can resist length extension attack .

# 6. Improvements to resist known attack

There are some ways to attack hash algorithm. People are interested in length extension attack and multicollision attack. HMAC can resist length extension attack. There are some ways can stop these attack.

## 6.1. Improvement one

Let H(.) is hash function. M is message, and $M^i$ are message block after padded, where $0 \leqslant i \leqslant L-1$.

$$hv = (H(M) \oplus H(C \oplus M^0 \oplus \ldots \oplus M^{L-1})) + H(M) \qquad (1)$$

Where C is constant . hv is hash value. (1) can stop extension attack and multicollision attack.

If H is keyd hash function, it is hard to get H(M), then it is hard to attack (1) by extension attack.

When the collision of H(M) is found, $H(C \oplus M^0 \oplus \ldots \oplus M^{L-1})$ maybe different. It is more harder to find multicollision of (1).

# 6.2. Improvement two

Let h(.) is compression function. M is message, and $M^i$ are message block after padded, $CV_i$ is chaining value, where 1⩽i ⩽ L-1. If the last chaining value is handled as fig 6 show:

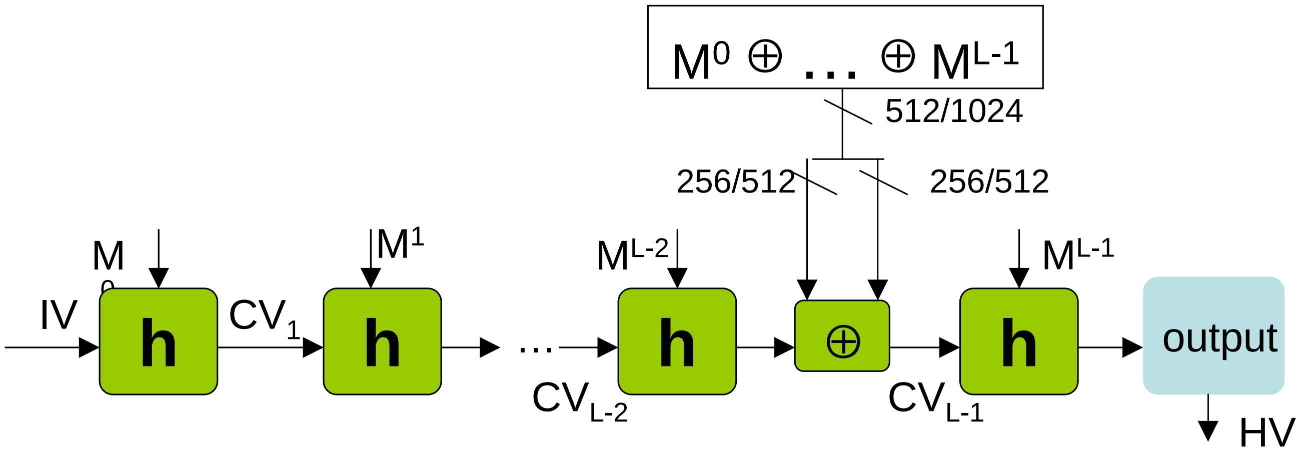


**Fig 7. Improvement two**

In improvement two as fig 7 show.

If H is keyd hash function, it is hard to get chaining value $CV_{L-1}$ and $CV_{L-2}$. And when attacker try to pad some bits, chaining value $CV_{L-1}$ maybe change. It is hard to attacked by length extension attack.

When it can find a different block data $M^i$ that do not change chaining value $CV_{L-2}$, chaining value $CV_{L-1}$ maybe different. It is hard to find multicollision by replace one block data $M^i$.

THANK YOU

END

January 12 . 2009