Design and Analysis Cryptographic Hash Function for The Next Generation

Her, Yong-Sork
Dept. Computer Science & Communication Engineering, KYUSHU University

Sakurai, Kouichi
Dept. Computer Science & Communication Engineering, KYUSHU University

https://doi.org/10.15017/3783

出版情報：Proc. of International Workshop on Information & Electrical Engineering. (1), pp.168-173, 2002-05. International Workshop on Information & Electrical Engineering
バージョン：

DESIGN AND ANALYSIS OF CRYPTOGRAPHIC HASH FUNCTION FOR THE NEXT GENERATION

Yong-Sork HER , Kouichi SAKURAI
Dept. Computer Science & Communication Engineering, KYUSHU University
6-10-1, Hakozaki, Fukuoka, 812-8581, Japan.
Tel : +81- 92-642-4050, Fax : +81- 92-632-5204,
Email: ysher@tcslab.csce.kyushu-u.ac.jp , sakurai@csce.kyushu-u.ac.jp

ABSTRACT

Hash functions play an important role in a branch of information secret. The hash algorithm provides the services of information security, authentication, integrity, non-reputation and so on. As the growth of computer technologies, the hash value has been become longer based on the complexity of calculation. It was known to be desirable that the output lengths of hash functions are more than 160 bits. The 1st edition of Hash Function Standard-160 of Korea, namely HAS-160, was published in 1998 and the revised edition was published in 2000. In this paper, we propose the three improved hash functions. First, we propose HAS-256, 384 and 512 that were based on SHA-256, 384 and 512. Second, we propose SHA-V using HAS-V proposed by P.J.Lee [12]. He proposed a new hash function with variable output length. Third, we propose the SHA-V with variable output length and another HAS-longer-version based on RIPEMD-256/320.

Keyword: *Informatics*, Cryptography, Information-Security. Hash function, Digital signature, Authentication, Integrity, HAS, SHA, MD4, RIPEMD, SEED

1. INTRODUCTION

Hash functions[1] take a message as input with an arbitrary length and produce an output referred to as a hash-code, hash-result, hash-value, or simply hash. More precisely, a hash function h maps bit strings of arbitrary finite length to strings of fixed length, say n bits.

Hash functions are used for data integrity in conjunction with digital signature schemes, where for several reasons a message is typically hashed first, and then the hash-value, as a representative of the message, is signed in place of the original message. <Table 1>[10] shows the efficiency of hash functions based on the block- cryptography algorithm DES.

A hash function satisfies the following conditions. First, a hash function maps an input with an arbitrary length to an output with a specific length. In this case, the length of output must be shorter than the length of input. Second, when y=h(x) is given, the inverse calculation of x must be computationally infeasible. It is said the 'one-wayness'. Third, a hash function h is strongly collision-free if it is computationally infeasible to find messages x and x' such that $x' \neq x$ and h(x')=h(x).

In 1990 Rivest proposed the cryptographic hash function MD4. MD4 [1]is a 128-bit hash function and consists of 3 rounds. After Merkle showed an attack on the first two rounds of MD4, den Boer and Bosselaers proposed an attack on the last two rounds of MD4, and Dobbertin finally demonstrated an attack for finding a collision for three-round MD4 [6]. So, It is undesirable to use MD4 as a hash function. MD5 was designed as a strengthened version of MD4 before MD4 collisions were found. MD5 has an input of 512 bits and output of 128 bits. It is found the attack method of MD5 that uses the collision on each different initial functions of two and one input that was called collision for random.

The important properties that a cryptographic hash function must satisfy are the following[1][10].

a) Preimage Resistance : for essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output,i.e., to find any preimage x' such that $h(x')=y$ when given any y for which a corresponding input is not known.

b) Second Preimage Resistance : it is computationally infeasible to find any second input which has the same output as any specified input,i.e.,given x, to find a 2nd-preimage $x' \neq x$ such that $h(x)=h(x')$.

c) Collision Resistance : it is computationally infeasible to find any two distinct inputs x, x' which hash to the same input,i.e., such that $h(x)=h(x')$.

One of the important roles in hash functions is digital signature. The goal of a practical hash function should be to achieve both preimage and collision resistance. In present, the length of output encourages at least 160 bits or more because of the security. This side can not to keep the safety of MD5, either. Several hash functions were designed as a strengthened version. For examples, there are SHS(Secure Hash Standard), HAVAL, SHA-1, REPEMD 160/256/320 and so on. In this paper, we will propose the improved hash functions.

<Table 1> The comparison of efficiency on Digital signature using hash functions

Classify	Time <seconds>	Memory
Not use of hash function	4,800	100Kbytes
Use of hash function	16	50.5Kbytes

(A size of transmission message: 50 Kbytes)

-Symbols

∧: Bitwise And operation
∨: Bitwise OR (“inclusive-OR”) operation
⊕: Bitwise XOR(“exclusive-OR”) operation
¬ : Bitwise complement operation
+ : Addition modulo 2^w
<< : Left-shift operation, where x<<n is obtained by discarding the left-most n bits of the word x and then padding the result with z zeroes on the right.
>> : Right-shift operation, where x>> n is obtained by discarding the right-most n bits of the word x and then padding the result with n zeroes on the left

2. HAS-160

HAS-160 (Hash Algorithm Standard) provides the methods to compress bit strings with arbitrary lengths into a hash code with fixed lengths (160 bits)[15]. This hash algorithm inputs messages of the random length with block unit of 512 bits. The length of output is 160 bits and compression function deals with block of 512-bit unit. It is composed of 4 rounds, 80 steps and 5 words of chaining variables. The number of variable message to apply in each step is 20 words. If it inputs the message *M* of the random length in this hash algorithm, *M* is made up of the multiple of 512-bit through attach processing and divided into the block M_i of 512 bits, ($o \le i \le t$).

- ***Initial values***

The initial hash values are as follows:
H_0 =67452301 , H_1 =efcdab89 , H_2 =98badcfe , H_3 =10325476, H_4 =c3d2e1f0

- ***Constants***

These constants are the integer parts of $2^{30}\sqrt{2}$, $2^{30}\sqrt{3}$ and $2^{30}\sqrt{5}$.
K_j = 00000000 ($0 \le j \le 19$: round 1)
K_j = 5a827999 ($20 \le j \le 39$: round 2)
K_j = 6ed9eba1 ($40 \le j \le 59$: round 3)
K_j = 8f1bbcdc ($60 \le j \le 79$: round 4)

- ***The ready of message valuable***

The message block M_i of each 512bits are transmitted to the 16 words, namely x[0], x[1], , , , x[15], by the transmit role of bit string-word string. Four messages are additional created with x[16], x[17], x[18], x[19] from 16 words.

<Table 2> Message ordering of HAS160

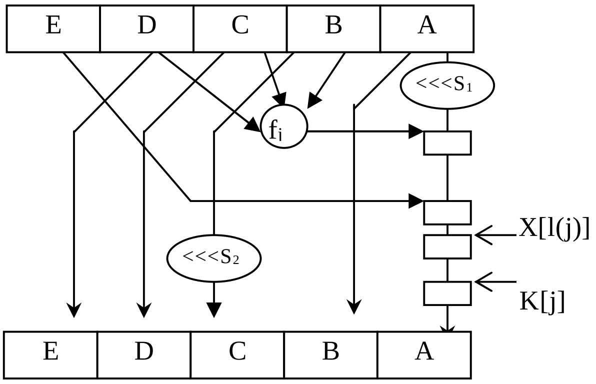
I	Round 1	Round2	Round 3	Round 4
0	X[18]	X[18]	X[18]	X[18]
1	X[0]	X[3]	X[12]	X[7]
2	X[1]	X[6]	X[5]	X[2]
3	X[2]	X[9]	X[14]	X[13]
4	X[3]	X[12]	X[7]	X[8]
5	X[19]	X[19]	X[19]	X[19]
6	X[4]	X[15]	X[0]	X[3]
7	X[5]	X[2]	X[9]	X[14]
8	X[6]	X[5]	X[2]	X[9]
9	X[7]	X[8]	X[11]	X[4]
10	X[16]	X[16]	X[16]	X[16]
11	X[8]	X[11]	X[4]	X[15]
12	X[9]	X[14]	X[13]	X[10]
13	X[10]	X[1]	X[6]	X[5]
14	X[11]	X[4]	X[15]	X[0]
15	X[17]	X[17]	X[17]	X[17]
16	X[12]	X[7]	X[8]	X[11]
17	X[13]	X[10]	X[1]	X[6]
18	X[14]	X[13]	X[10]	X[1]
19	X[15]	X[0]	X[3]	X[12]

- ***Boolean functions***

The three boolean functions were used in this algorithm. These boolean functions using each jth step computation are following:
$f_j(x, y, z) = (x \wedge y) \vee (\sim x \wedge z)$ ($0 \le j \le 19$:round 1)
$f_j(x, y, z) = x \oplus y \oplus z$
($20 \le j \le 39$, $40 \le j \le 59$: round 2, 4)
$f_j(x, y, z) = y \oplus (x \vee \sim z)$ ($40 \le j \le 59$: round 3)
(∼:NOT, ∧:AND, ∨:OR, ⊕ :XOR)

- ***Step computations***

The step computations of each jth are as follows [1]:
$T \leftarrow A^{<<S1(j)} + f_j(B,C,D) + E + X[l(I)] + K_j$; $E \leftarrow D$; $D \leftarrow C$; $C \leftarrow B^{<<S2(j)}$; $B \leftarrow A$; $A \leftarrow T$;



<Figure1> Step computation

3. The proposes of HAS-256, 384 and 512

3.1 HAS-256

HAS-256 operates on the methods of MD4, MD5, SHA-1 and SHA-256[11]. The HAS-256 compression function operates on a 512-bit message block and a 256-bit intermediate hash value. It is the essential 256-bit block cipher algorithm that encrypts the intermediate hash value using the message block as key.

- *Padding message*

For HAS-256, the padded message is parsed into N 512-bit blocks; $M^{(1)}$, $M^{(2)}$, ⋯ , $M^{(N)}$. For interoperable implementations involving byte-to-word conversions, this algorithm uses the little-endian. In little-endian architecture, the byte with the lowest memory address is the least significant byte; $W= 2^{24}B_4+ 2^{16}B_3+2^8B_2+B_1$.

- *Initial values*

For HAS-256, the initial hash value $H^{(0)}$ shall consist of the following eight 32-bit words.

$H_0^{(0)}$ = 6a09e667, $H_1^{(0)}$ = bb67ae85,
$H_2^{(0)}$ = 3c6ef372, $H_3^{(0)}$ = a54ff53a
$H_4^{(0)}$ = 510e527f, $H_5^{(0)}$ = 9b05688c,
$H_6^{(0)}$ = 1f83d9ab, $H_7^{(0)}$ = 5be0cd19.

- *Functions*

$Ch(x,y,z) = (x\wedge y) \oplus (\neg x\wedge z)$
$Maj(x,y,z) = f_t (x, y, z)=(x\wedge y) \oplus (x\wedge z) \oplus (y\wedge z)$
$\Sigma_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$
$\Sigma_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$
$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$
$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$

- *Constants*

HAS-256 constants are the same with SHA-256. That is, HAS-256 uses a sequence of 64 constants 32-bit words, $K_0^{\{256\}}$, $K_1^{\{256\}}$, ⋯, $K_{63}^{\{256\}}$. These words represent the first 32-bit of the fractional parts of the cube roots of the first 64 prime numbers.

3.2 HAS-384 and HAS-512

- *HAS-384 and HAS-512 functions*

HAS-384 and HAS-512 use each six logical functions, where each function operates on 64-bit words which are represented as x, y, and z. The result of each function is a new 64-bit word. HAS-512 uses a variant of HAS-256 that operates on eight 64-bit words.

The HAS-512 compression function operates on a 1024-bit messages block and a 512-bit intermediate hash value. It is the essential 512-bit block cipher algorithm that encrypts the intermediate hash value using the message block as key.

$Ch(x,y,z) = (x\wedge y) \oplus (\neg x\wedge z)$
$Maj(x,y,z)=f_t(x, y, z) = (x\wedge y) \oplus (x\wedge z) \oplus (y\wedge z)$
$\Sigma_0^{\{512\}}(x) = ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x)$
$\Sigma_1^{\{512\}}(x) = ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x)$
$\sigma_0^{\{512\}}(x) = ROTR^1(x) \oplus ROTR^8(x) \oplus SHR^7(x)$
$\sigma_1^{\{512\}}(x) = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^6(x)$

- *HAS-384 and HAS-512 Constants*

HAS-384 and HAS-512 use the same sequence of eighty constant 64-bit words, $K_0^{\{512\}}$, $K_1^{\{512\}}$, ⋯ ,$K_{79}^{\{512\}}$. These words represent the first sixty-four bits, which are the fractional parts of the cube roots of the first eighty prime numbers. These constant words are as like SHA-384 and SHA-512.

- *Initial values of HAS-384 and HAS-512*

For HAS-384 and HAS-512, the initial hash value $H^{(0)}$ shall consist of the following eight 64-bit words.

①Initial value of HAS-384
$H_0^{(0)}$ = cbbb9d5dc1059ed8, $H_1^{(0)}$ = 629a292a367cd507,
$H_2^{(0)}$ = 9159015a3070dd17, $H_3^{(0)}$ = 152fecd8f70e5939,
$H_4^{(0)}$ = 67332667ffc00b31, $H_5^{(0)}$ = 8eb44a8768581511,
$H_6^{(0)}$ = db0c2e0d64f98fa7, $H_7^{(0)}$ = 47b5481dbefa4fa4
② Initial value of HAS-512
$H_0^{(0)}$= 6a09e667f3bcc908, $H_1^{(0)}$= bb67ae8584caa73b,
$H_2^{(0)}$ = 3c6ef372fe94f82b, $H_3^{(0)}$ = a54ff53a5f1d36f1,
$H_4^{(0)}$ = 510e527fade682d1, $H_5^{(0)}$ = 9b05688c2b3e6c1f,
$H_6^{(0)}$ = 1f83d9abfb41bd6b, $H_7^{(0)}$ = 5be0cd19137e2179

3.3 Security of HAS-256,384,512

The security of hash functions has been studied. The length of the hash-code is an important factor to connect directly to the security of the hash function.

① A round is added. This means the difficulty to attack the each round.
② Each step now has a unique additive constant.

③ Add the result of the previous step to each step.
④ The order in which message sub-blocks are in the each round is changed.
⑤ The left circular shift amounts in each round have been approximately optimized, to yield a faster avalanche effect.

4. RIPEMD-160

4.1 Introduction of RIPEMD

The main contribution of MD4 is first cryptographic hash function. MD4 was made optimally to use of structure of current 32-bit processors [5]. The design of MD4 represented an uncomfortable compromise between security and speed [14]. As a consequence, the more conservatively designed MD5 has always been recommended for using instead of MD4. It was intended to be used as a secure replacement for the 128-bit hash functions MD4, MD5 and RIPEMD. MD4 and MD5 were developed by Ron Rivest for RSA Data Security, while RIPEMD was developed in the framework of the EU project RIPE[1].

4.2 RIPEMD-160

RIPEMD-160 [2][5] is a 160-bit cryptographic hash function designed by Hans Dobbertin, Antoon Bosselaers and Bart Prenell. There are two good reasons to consider such a replacement.

- A 128-bit hash result does not offer sufficient protection anymore.
- In the first half of 1995, Hans Dobbertin found collisions for a version of RIPEMD restricted to two rounds out of three.

RIPEMD-160 is a strengthened version of RIPEMD with a 160-bit hash result. The bit-size of the hash-result and chaining variable for RIPEMD-160 are increased to five 32-bit words (160 bits), the number of round is increased from three to five, and the two lines are made more different. The results of parameters are as following.

- *Operations in one step*

$A = (A + f(B,C,D) + X + K)^{\ll S} + E$, and $C = C^{\ll 10}$,
Here $^{\ll S}$ denotes cyclic shift (rotation) over a positions.

- *Ordering of the message words*

Take the following permutation *p*:

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

Line	Round1	Round2	Round3	Round4	Round5
Left	*Id*	*p*	p^2	p^3	p^4
right	π	$p\pi$	$p^2\pi$	$p^3\pi$	$p^4\pi$

- *Boolean functions*

Boolean functions are as following:

$f_1(x, y, z) = x \oplus y \oplus z$
$f_2(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$
$f_3(x, y, z) = (x \vee \neg y) \oplus z$
$f_4(x, y, z) = (x \wedge z) \vee (y \wedge \neg z)$
$f_5(x, y, z) = x \oplus (y \vee \neg z)$

These Boolean functions are applied as following.

Line	Round1	Round2	Round3	Round4	Round5
Left	f_1	f_2	F_3	f_4	f_5
Right	f_5	f_4	F_3	f_2	f_1

- *Constants*

Take the integer parts of the following numbers.

Line	Round1	Round2	Round3	Round4	Round5
Left	0	$2^{30} \cdot \sqrt{2}$	$2^{30} \cdot \sqrt{3}$	$2^{30} \cdot \sqrt{5}$	$2^{30} \cdot \sqrt{7}$
right	$2^{30} \cdot \sqrt{7}$	$2^{30} \cdot \sqrt{5}$	$2^{30} \cdot \sqrt{3}$	$2^{30} \cdot \sqrt{7}$	0

The basic design philosophy of RIPEMD was to have two parallel iterations; the two main improvements are that the number of rounds is increased from three to five.

4.3 Optional Extensions to 256 and 320 bit Hash-Results

RIPEMD-256 and RIPEMD-320 are optional extension of RIPEMD-128 and RIPEMD-160, and are intended for applications of hash functions that require a longer hash result without needing high security level.
Some applications of hash functions require a longer hash-result without needing high security level. A straightforward way to achieve this would be to use two parallels instances of the same hash function with different initial values. An extension of MD4, which yields a 256-bit hash-result by running two parallels instances of MD4 that differs only in the initial values and the constants in the second and third round, was proposed. After every application of the compression function, the value of the register A is interchanged between the two chains.

5. HAS-longer-version based on RIPEMD-256 and RIPEMD-32

5.1 HAS-longer-version based on RIPEMD-256

RIPEMD-256 is an iterative hash function that operates

[1] RACE Integrity Primitives Evaluation, 1988-1992

on 32-bit words. The round function takes as input an 8-word chaining variable and a 16-word message block, and maps this to a new chaining variable. All operations are defined as 32-bit words. The padding is identical with that of HAS.

- *Boolean functions*

f_J (x, y, z) = x ⊕ y ⊕ z (0 ≤ j ≤15),
f_J (x, y, z) = (x∧y)∨(¬x∧z) (16 ≤ j ≤31),
f_J (x, y, z) = (x∨¬y) ⊕z (32 ≤ j ≤47),
f_J (x, y, z)= (x∧z)∨(y∧¬z) (48 ≤ j ≤63)

- *Constants*

K(j):000000 (0 ≤j ≤15), K(j):5a827999 (16 ≤j ≤31), K(j):6ed9eba1(32 ≤j ≤47), K(j):8f1bbcdc (48 ≤j ≤63), K'(j):50a28be6(0 ≤j ≤15),K'(j):5c4dd124(16 ≤j ≤31), K'(j):6d703ef3 (32 ≤j ≤47), K'(j):00000000(48 ≤j ≤63)

- *Initial values*

h0: 67452301, h1: efcdab89, h2: 98badcfe, h3:10325476, h4:76543210, h5: fedcba98, h6:89abcdef, h7:01234567

5.2 HAS-longer-version based on RIPEMD-320

The round function takes as input a 10-word chaining variable and a 16-word message block, and maps this to a new chaining variable. The padding is identical with that of HAS.

- *Boolean functions*

f_J (x, y, z) = x ⊕ y ⊕ z (0≤j ≤15),
f_J (x, y, z) = (x∧y)∨(¬x∧z) (16 ≤j ≤31),
f_J (x, y, z) = (x∨¬y⊕z (32≤j ≤47),
f_J (x, y, z)= (x∧z)∨(y∧¬z) (48≤j ≤63)
f_J (x, y, z) = x ⊕ (y∨¬z) (64≤j ≤79)

- *Constants*

K(j):000000 (0 ≤j ≤15), K(j):5a827999 (16 ≤j ≤31), K(j):6ed9eba1 (32 ≤j ≤47),K(j):8f1bbcdc (48 ≤j ≤63), K(j):a953fd4e(64 ≤j ≤79),K'(j):50a28be6 (0 ≤j ≤15) , K'(j):5c4dd124(16 ≤j ≤31), K'(j):6d703ef3 (32 ≤j ≤47) , K'(j):7a6d76e9, (48 ≤j ≤63), K'(j):00000000, (48 ≤j ≤63)

- *Initial values*

h0:67452301, h1:efcdab89, h2:98badcfe, h3: 10325476, h4:c3d2e1f0, h5:76543210, h6: fedcba98, h7:89abcdef, h8:01234567, h9:3c2d1e0f

6. HAS-V

HAS-V[12] was proposed to meet the needs of various security levels desired among different applications. The length of the hash-code is an important factor directly connected to the security of the hash function. KCDSA (Korea Certificate-based Digital Signature Algorithm) is an example of a cryptographic application where a variable length of hash-code is needed. There exists an optional extension of RIPEMD-128 and RIPEMD-160 to produce 256-bit and 320-bit hash-code. However, these methods do not provide any increase in security level, but merely an increase in the length of the hash-code. This gives a clear motivation to design a new hash function with variable length hash-code, which is both efficient and secure.

- +: addition of words, i.e., addition by modulo-2^{32}
- $X^{<<s}$: the circular left shift of X by *s* bit positions
- ¬ : the bitwise complement operation
- ∧,∨, ⊕ : the bitwise OR, AND, and XOR operation (X ∧Y is also denoted as XY for simplicity)

<Table 3> Characteristics of HAS-V

Length of Input Block (bits)	1024
Length of Output (bits)	128- 320
Number of Rounds	10
Number of Chaining Variables	10
Number of Steps	200

- *Initial values*

A: 67452301, B: efcdab89, C: 98badcfe, D: 10325476
E: c3d2e1f0, F: 8796a5b4, G: 4b5a6978, H: 0f1e2d3c
I : a0b1c2d3, J : 68794e5f

- *Step operation*

T←$A^{<<s}$ + f(B,C,D,E) +X+K; E←D;D←C;
C←$B^{<<30}$; B←A; A←T;

- *Boolean Function*

f_0 (x,y,z,u) = xy ⊕ ¬xz ⊕ yu⊕ zu,
f_1 (x,y,z,u) = xz ⊕ y ⊕ u
f_2 (x,y,z,u) = xy ⊕ ¬xu ⊕ z,
f_3 (x,y,z,u) = x⊕yz⊕u (=f_1 (y,x,z,u))
f_4 (x,y,z,u) = ¬xy⊕ xz ⊕ yu ⊕ zu (=f_0 (x,z,y,u))

- *Constants*

K_0:000000, K_1:5a827999, K_2:6ed9eba1,
K_3: 8f1bbcdc, K_4:a953fd4e

7. The propose of SHA-V

We propose a hash function with variable output length based on SHA, namely SHA-V. The basic structure of the compression function is same as HAS-V[12]. That is, it is two parallel lines, denoted as the X-line and the

Y-line and consists of 100 steps each. Each line is composed of 5 rounds, where each round consists of 20 steps, and maintains 5 words of chaining variables in the X-line and Y-line after each round. The message words using in the compression function are 32 words of the input messages.

- *Initial values*

$H_0^{(0)}$ = 67452301, $H_1^{(0)}$ = efcdab89,
$H_2^{(0)}$ = 98badcfe, $H_3^{(0)}$ = 10325476,
$H_4^{(0)}$ = c3d2e1f0, $H_5^{(0)}$ = 8796a5b4,
$H_6^{(0)}$ = 4b5a6978, $H_7^{(0)}$ = 0f1e2d3c,
$H_8^{(0)}$ = a0b1c2d3, $H_9^{(0)}$ = 68794e5f

- *Boolean function*

<Table 4> Order of Boolean function

Line	Round1	Round2	Round3	Round4	Round5
X	f_0	f_1	f_2	f_3	f_4
Y	f_4	f_3	f_2	f_1	f_0

f_j (x,y,z,u) = xy ⊕ ¬xz⊕yu⊕zu (0 ≤ j ≤19),
f_j (x,y,z,u) = xz⊕y⊕u (20 ≤ j ≤39),
f_j (x,y,z,u) = xy⊕¬xu⊕z (40 ≤ j ≤59),
f_j (x,y,z,u) = x⊕yz⊕u (60 ≤ j ≤79),
f_j (x,y,z,u) = ¬xy⊕xz⊕yu⊕zu (80 ≤ j ≤99),
g_j (x,y,z,u) = ¬xy⊕xz⊕yu⊕zu (0 ≤ j ≤19),
g_j (x,y,z,u) = x⊕yz⊕u (20 ≤ j ≤39),
g_j (x,y,z,u) = xy⊕¬xu⊕z (40 ≤ j ≤59),
g_j (x,y,z,u) = xz⊕y⊕u (60 ≤ j ≤79),
g_j (x,y,z,u) = xy⊕¬xz⊕yu⊕zu (80 ≤ j ≤99)

- Constants

K_j:000000, K'_j:a953fd4e, (0 ≤ j ≤19),
K_j:5a827999, K'_j: 8f1bbcdc, (20 ≤ j ≤39),
K_j:6ed9eba1, K'_j:000000, (40 ≤ j ≤59),
K_j: 8f1bbcdc, K'_j:5a827999, (60 ≤ j ≤79),
K_j:a953fd4e, K'_j:6ed9eba1, (80 ≤ j ≤99)

8. Conclusion

We proposed HAS-256, 384 and 512 which were based on SHA-256, 384 and 512. HAS functions have the structure of little-endian and are suited for a systematization of Intel 80XXX. On the other hand, SHA functions choose the structure of big-endian. Generally, the processor of big-endian structure is faster than little-endian's [4]. Therefore, we have to design the algorithm to fit the structure of little-endian. It exists always as the pair of collision because an input has a lot of data than the output in the hash function. Because an input has a lot of numbers, the existing probability of collision pairs is high. That is, the safety of hash functions depends on the difficulty to find the collision pairs. Also, we propose the SHA-V based on the HAS-V. This design is expected to be used to the several cryptographic applications because it has the various outputs of hash code.

9. References

[1] A.J. Menzs, P.C. Van Oorshot, S.A.Vanstone "Handbook of Applied Cryptography" CRC Press, 1997

[2] Antoon Bosselaers " The Hash Function RIPEMD 160" http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.html

[3] B.Schneier "Applied Cryptography" WILEY, Vol 2, 1996

[4] C.C.Park "Cryptography Theory and Security" Deyung Press, 1999

[5] H.Dobbertin,.Bosselaers, B.Preneel " RIPEMD-160 : A Strengened Version of RIPEMD " http://www.esat.kulenven.ac.be/~cosicart/pdf/

[6] H.KUWAKADO, H. TANAKA " New Algorithm for Finding Preimage in a Reduced Version of the MD4 Compression Function" IEICE TRANS, Fundamentals, Vol.E83-A. No.1, Jan, 2000

[7] Korea Information Security Agency " A Design and Analysis of SEED" Dec, 1999

[8] M.J.B.Robshaw "On Recent Results for MD2, MD4, MD5" April, 1996

[9] M.S. Lee "Modern Cryptography" KyoU Press, 1999

[10]National Security Research Institute " Modern Cryptology" Kyungmoon, 2000

[11]NIST " Descriptions of SHA-256, SHA-384, and SHA-512" http://csr.nist.gov/cryptval/shs.html

[12]N.K.Park, J.H.Hwang, P.J.Lee " HAS-V : A New Hash Function with Variable Output Length" SAC2000, LNCS2012, Springer-Verlag, 2001

[13]P.Sarkar, P.J.Schellenberg "A Paralleizable Design Principle for Cryptographic Hash Function " http://eprint.iacr.org/2002/031

[14]RSA Laboratories " Bulletin" Number 4, Nov.12, 1996, http://www.ras.com

[15]TTA Standard " Hash Function Standard - Part 2: Hash Function Algorithm Standard (HAS-160) " TTA,KO-12.0011/R1, Dec

[16]U.S Department of Commerce, NIST " Secure Hash Standard" FIPS PUB 180-1, Apr, 1995