TTA Standard
TTAS.KO-12.0011/R1

Enacted on : 20 Dec. 2000

HASH FUNCTION STANDARD - PART 2 : HASH FUNCTION ALGORITHM STANDARD(HAS-160)

HASH FUNCTION STANDARD - PART2:HASH FUNCTION ALGORITHM STANDARD<HAS160>

Telecommunications Technology Association

TTA Standard

HASH FUNCTION STANDARD - PART 2 : HASH FUNCTION ALGORITHM STANDARD(HAS-160)

Telecommunications
Technology Association

Preface

1. Purpose of Standard

This standard specifies HAS-160(Hash Algorithm Standard) that provides methods to compress bit strings with arbitrary lengths into a hash code with fixed lengths(160 bits).

2. Recommended Recommendations and/or Standards

2.1 International Standards

ISO/IEC 10118-1(1994) : Information Technology-Security techniques
Hash functions - Part 1 General

ISO/IEC 10118-2(1994) : Information Technology-Security techniques
Hash functions - Part 2 : Hash functions
using an n-bit block cipher algorithm

ISO/IEC 10118-3(1996) : Information Technology-Security techniques
Hash functions - Part 3 Dedicated hash function

ISO/IEC 10118-4(1996) : Information Technology-Security techniques
Hash functions - Part 4 Hash functions
using modular arithmetic

2.2 Domestic Standards : KIS 133 : Hash function Standard / Framework

2.3 Other Standards : None

3. Relationship to International Standards

3.1 The Relationship of international standards : None

3.2 Additional requirements to the international standard : None

4. The statement of intellectual Property Rights : None

5. The Statement of conformance testing and certification : None

6. The history of standard

Edition	Issue Date	Contents
The 1st edition	1998. 10. 27.	Established
The 2nd edition	2000. 12. 20.	Revised

CONTENTS

HASH FUNCTION STANDARD - PART 2: HASH FUNCTION ALGORITHM STANDARD(HAS-160)

1. Introduction

Hash algorithm, as algorithm that compresses a certain length of bit row into a fixed output value, satisfies the following features;

1. It is impossible to calculate input value for the given output in view of calculation.

2. It is impossible to find another input which will generate the same output for the given input in view of calculation.
   Most hash algorithm used in crypto application is required to have not only these two features but also much stronger collision resistance than these. That is,

3. It is impossible to find two different input messages that generate the same output in view of calculation.(collision-resistance).
   The collision resistance of crypto hash algorithm is an essential requirement for non-repudiation service to prevent falsification by the third party but sender in digital signature.

Hash algorithm can be divided into the hash algorithm based on block cipher algorithm such as DES and the leased hash algorithm. The algorithm using block cipher algorithm has an advantage that it can use already implemented block cipher algorithm but the speed of most block cipher algorithm is not fast and in case of the hash algorithm that uses basic function, its speed is much lower than block cipher algorithm, therefore today, in most applications, leased hash algorithm is mainly used.

Most leased has algorithm is calculated through add. division, loop operation process. A certain length of message X to make it multiple of input unit is added and then divided into t number of input block(($X_1, \cdots, X_t$).
Hash code is calculated by repeatedly applying to compression function after initializing chain variable into a given initialization value(IV). its process is described as followings.

$$H_0 = IV,$$

$$H_i = f(H_{i-1}, X_i),\ 1 \le i \le t,$$

$$h(X) = H_t$$

Here, f is compression function of h, H_i is calculation value in the middle of stage i-1 and stage i.

2. Scope

The hash algorithm defined in this standard can be used for digital signature that provides information protect service such as authentication, integrity and non-repudiation in telecommunication network environment. Or, according to this standard, it can be used for checking whether implementation results are proper when developing telecommunication equipments and applications.

3. References

KIS 133 : Hash Function Standard/Basic Structure

4. Definitions

Definitions below are used in this standard respecting those in the governed standard.

A. Adding : Add additional bits to a certain data row

B. Digital Signature : Data that is generated cryptologically, or additional data to prove data originating source and integrity, and to prevent falsification for data recipient.

C. Collision resistance hash algorithm : hash algorithm whose feature is represented by the fact that it is impossible to find two different input data rows that have the same output(Hash code) in view of calculation.

D. Hash Code : Output bit row of hash algorithm

E. Hash algorithm : Function that makes data row match to a fixed length of hash code satisfying two features below.

1) It is impossible to find data rows generating the same hash code with a give hash code in view of calculation.

2) It is impossible to find another data rows generating the same hash code with a give hash code in view of calculation.

F. Word : 32 bit row

G. Block : Input bit number of compression function in hash algorithm(512bit)

5. Notations and Conventions

5.1 General symbol and notation

Symbols notated in English Capital letters are not negative numbers but integer values or corresponding bit rows. Symbols used in this standard are as followings.

+ : Addition operation in which the value between words is 2^{32} .

$X^{<<s}$: Operation to circulate and move X to the left as far as s bit.

¬, ∨, ∧, ⊕ : NOT, OR, AND, XOR operation by bit

H_0 ,H_1 , H_2 , H_3 , H_4 : Chain variable

K_j : 32 bit constant which is used in operation of stage j

f_j : Boolean function which is used in operation of stage j

$s_1(j)$, $s_2(j)$: Amount(bit number) circulated and moved which is used in operation of stage j

A += B : Input the value of A + B to A for a certain A and B

5.2 Conversion between bit row and word

In internal operation of standard hash algorithm, operation between 32 bit of integers. That is, a certain length of bit row converts into 32 bit word row, its result is converted into bit row of 160 bit and then, hash code is output. Therefore, little-endian conversion is needed between bit row and word row. In this standard, the following little-endian conversion is used.

A. Conversion between bit row of 32 bit and word

Bit row of 32 bit is considered as word row of 4 bite and the first bite becomes the lowest level bite. For example, bit row

10101101 01101011 11001001 10101110 = ad6bc9ae

becomes W = aec96bad in 32 bit word. This is the same with the result of operation in which the 4 bit word row above is type cast into unsigned long in little-endian computer.

B. When converting a certain length of bit row into 32 bit word row, consider this bit row as bite row and the first 4 bite as the first word and repeat this conversion process converting the second bite into the second word and so on. For example, bit row

10101101 01101011 11001001 10101110 00111111 01011001 01000110
= ad6bc9ae3f5946

becomes aec96bad 0046593f in 32 bit word row.

C. When converting 32 bit word row to bit row, reverse the processes of A and B.

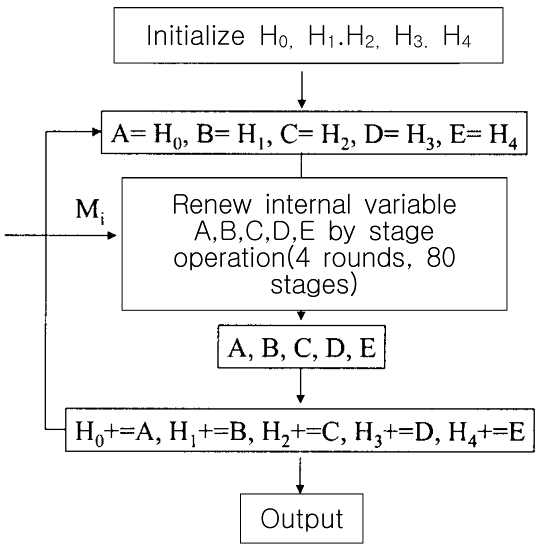
D. 64 bit integer $Z=2^{32}X + Y(0 \leq X, Y<2^{32})$ is converted into 32 bit word row (Y, X). That is, low level 32 bit becomes the first word.

6. Techniques of HAS-160

6.1. Overall Structure

Standard hash algorithm generates 160 bit output by processing a certain length of input message into 512 bit block unit. Compression function consists of all 4 rounds and 80 stages and chain variables calculating hash code are 5. Also, the number of message variable to be applied to each round becomes 20 including 16 words created from 512 bit input block and 4 words additionally created from those.

When message M is input, standard hash algorithm makes this M in multiple of 5123 bit through addition process and then, divides it by 512bit block M_i ($0 \le i < t$). After each block is compressed through such a process as seen in the picture below and done final block process, chain variables H_0 ,H_1 , H_2 , H_3 , H_4 are converted into bit rows. And the converted ones become hash codes.



6.2. Input Block Length and Addition

Input message is processed in 512 bit unit. After filling the needed numbers of “0” next to “1” to make its block length 448 bit for the last message block, enter the integer value calculated the previous message length before adding in 2^{64} method for the last 64 bit. For example, suppose that input message is given in the following bit row.

10100010 00111001 11100101 01101011 11001001 10001010 10011101

Since the length of this bit row is 56, make it in 448 bit by adding 391 numbers of 0 and enter 56 = 111000 for the rest of 64 bit. Therefore, from the result, 32 bit word row of 512 bit is given as following;

6be539a2 809d8ac9 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000038 00000000

6.3 Initial Value

In this standard hash algorithm, the values used in initialization of chain variable are notated in hexadecimal as below.

H_0 = 67452301
H_1 = efcdab89
H_2 = 98badcfe
H_3 = 10325476
H_4 = c3d2e1f0

6.4 Constant

The constants to be used in operations on each stage are notated in hexadecimal as below. These constants take irrational number, integer part of $2^{30}\sqrt{2}$, $2^{30}\sqrt{3}$, $2^{30}\sqrt{5}$.

K_j = 00000000 (0≤j≤19: Round 1)

K_j = 5a827999 (20≤j≤39: Round 2) $\lfloor 2^{30}\sqrt{2} \rfloor$

K_j = 6ed9eba1 (40≤j≤59: Round 3) $\lfloor 2^{30}\sqrt{3} \rfloor$

K_j = 8f1bbcdc (60≤j≤79: Round 4) $\lfloor 2^{30}\sqrt{5} \rfloor$

6.5. Preparing Message Variable

Each 512bit message block M_i can be converted into 16 words-X[0], X[1]…, X[15]- according to conversion rules by bits and words, and additional 4 message variables ()- X[16], X[17], X[18], X[19] are created from 16 words. The order in which message variables to be input to stage operations of each round is given as the table below. 4 additional variables X[16], X[17], X[18], X[19] are used in 10th, 15th, 0, 5th stage operation of each round and their values are calculated with XOR values of message variables used in 0-4, 6-9, 11-14, 16-19 stage operations of each round. For example, X[16], X[17], X[18], X[19] of the second round are calculated as followings;

$$
\begin{aligned}
X[16] &= X[3] \oplus X[6] \oplus X[9] \oplus X[12] \\
X[17] &= X[15] \oplus X[2] \oplus X[5] \oplus X[8] \\
X[18] &= X[11] \oplus X[14] \oplus X[1] \oplus X[4] \\
X[19] &= X[7] \oplus X[10] \oplus X[13] \oplus X[0]
\end{aligned}
$$

i	round 1	round 2	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]

For the sake of convenience, let's notate the message variables to be used in stage operation j as X$[l(j)]$. Then according to the above table, $l(j)$ is as follows.

$l(j)$ = 18, 0, 1, 2, 3,19, 4, 5, 6, 7,16, 8, 9,10,11,17,12,13,14,15 ($0 \le j \le 19$)
$l(j)$ = 18, 3, 6, 9,12,19,15, 2, 5, 8,16,11,14, 1, 4,17, 7,10,13, 0 ($20 \le j \le 39$)
$l(j)$ = 18,12, 5,14, 7,19, 0, 9, 2,11,16, 4,13, 6,15,17, 8, 1,10, 3 ($40 \le j \le 59$)
$l(j)$ = 18, 7, 2,13, 8,19, 3,14, 9, 4,16,15,10, 5, 0,17,11, 6, 1,12 ($60 \le j \le 79$)

6.6. Boolean Function

Three boolean functions is used, and the boolean functions used in each stage operation j are as follows.

$f_j(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$ (0≤j≤19: Round 1)

$f_j(x, y, z) = x \oplus y \oplus z$ (20≤j≤39, 60≤j≤79: Round 2, 4)

$f_j(x, y, z) = y \oplus (x \vee \neg z)$ (40≤j≤59: Round 3)

That is, the same boolean function is used in each round, and equal boolean function is used in round 2 and 4.

6.7. Amount of Circular Movement

The circular movement amount s_1 and s_2 needed in each stage operation j is defined as blow.

$$s_1(j) = 5, 11, 7, 15, 6, 13, 8, 14, 7, 12, 9, 11, 8, 15, 6, 12, 9, 14, 5, 13$$
$$(0 \le j \le 19,\ 20 \le j \le 39,\ 40 \le j \le 59,\ 60 \le j \le 79)$$

$$s_2(j) = 10 \quad (0 \le j \le 19)$$
$$s_2(j) = 17 \quad (20 \le j \le 39)$$
$$s_2(j) = 25 \quad (40 \le j \le 59)$$
$$s_2(j) = 30 \quad (60 \le j \le 79)$$

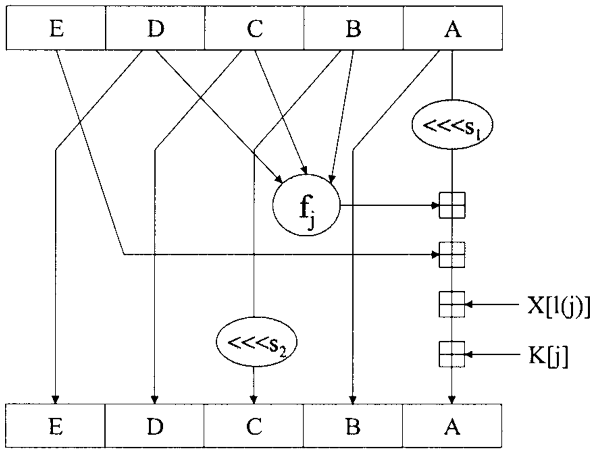
Circular movement amount s_1 is applied in the same order in each round and circular movement amount s_2 is applied with the same value in each round.

6.8. Stage Operation

The stage operations of each j are defined as blow. (Refer to the picture below).

$$T \leftarrow A^{<<s_1(j)} + f_j(B, C, D) + E + X[l(j)] + K_j;$$

$$E \leftarrow D;\ D \leftarrow C;\ C \leftarrow B^{<<s_2(j)};\ B \leftarrow A;\ A \leftarrow T;$$



6.9. Renewal Process of Chain Variable

After completing compression function operations of 4 rounds and 80 stages, renew chain variables $H_0 \sim H_4$ by adding those results A, B, C, D, E to chain variables $H_0 \sim H_4$.

H_0 += A, H_1 += B, H_2 += C, H_3 += D, H_4 += E

6.10. Hash Code Output

When all 512 bit message blocks are processes, chain variables H_0 ~ H_4 are converted into word row according to the conversion rules by words and bits described in chapter 5.2 and then, output in hash code. When each variable H_i is given as H_i = h_{i3} h_{i2} h_{i1} h_{i0} (h_{ij} is bit row of 8 bit), hash code becomes as the bit row blow.

$$h_{00}\,h_{01}\,h_{02}\,h_{03}\,h_{10}\,h_{11}\,h_{12}\,h_{13}\,h_{20}\,h_{21}\,h_{22}\,h_{23}\,h_{30}\,h_{31}\,h_{32}\,h_{33}\,h_{40}\,h_{41}\,h_{42}\,h_{43}$$

7. Design Criteria for HAS-160

This hash algorithm standard is designed based on design principles and analyses for existing NID4 related hash algorithm. Compression function's overall structure, addition, initial value and constant are similar to other hash algorithm. Here, several design standards are described based on special features of hash algorithm as below.

7.1. Message Variable Process

Since most of existing attacks are caused by simple applications of message variables, it is designed for one message variable to affect on various stages. The process in SHA-1, where a message word is created by using the total of 4 already-input message words, which will be input in each stage, is a good example to show this but, since too complicate method may affect on its effectiveness, it is designed to ensure fast implement as well as sufficient application frequencies of message variables.

Not only 4 additional message variables are created by doing XOR different 4 message variables out of 16 message variables in each round but also each message application becomes different by differentiating composition of the message variables done XOR to improve the simplicity. This standard hash algorithm refers the application order of message variables to the design standard of RIPEMD-160 and makes each message variable applied in a fixed interval with the same frequencies. Especially, additional message variables earned by XOR values of 4 message variables are regularly arranged in “{X[16], X[17], X[18], X[19]}” application order to make them applied in a sufficient interval(more than 5 stages) with 4 message variables to be done XOR. In addition, the same message variables are not applied next to each other in each round. Different value of circular movement amount is applied in each round.

7.2. Stage Operation

Stage operations basically use SHA-1. This is because SHA-1 stage operation method shows much stronger resistance to existing well-known attacks and it also has many advantages in paralleling or pipeline processing.

7.3. Boolean Function

The Boolean function used in stage operation uses the functions which use the numbers less than 3 as unit operation numbers for calculation among 3-variable Boolean functions widely used in existing MD4 related hash algorithm (The first Boolean function can be calculated as following: $(x \wedge y) \vee (\neg x \wedge z) = (x \wedge (y \vee z)) \vee z)$.) For reference, unit operation numbers represent the needed operation numbers in Boolean function when $\neg, \vee, \wedge, \oplus$ is considered as the same unit operation. The Boolean function used in Round 1 satisfies SAC(Strict Avalanche Criterion) and has 2, the biggest non-linear degree of 3-variable Boolean functions. The Boolean function used in Round 3 does not satisfy SAC but has the biggest value 2. Since the Boolean functions used in Round 2 and 4 have the lowest calculation values, they are double used among other Boolean functions with high non-linear degrees.

7.4. Conversion between word row and integer

This standard hash algorithm has 32 bit processor as its infra structure and is designed based on little-endian structure computer. Therefore, in big-endian structure such as most workstations, the bit order of message words should be reversed. This is according to the development trend of hash algorithm and this is because in general most MD4 related hash algorithms are designed in 32 bit little-endian structure.

Annex 1. Pseudo-code of HAS-160

1.1 Definition

Suppose that the message to be hashed consists of t numbers of 512bit block $\{M_i\}$ $(0 \le i < t)$(Here, the last message block is a 512 bit block created by adding 448 bit message and 64 bit message length through addition process.) Each message block of stage i consists of 16 words $M_i = \{X_i[j]\}$ $(0 \le j < 16)$ converted into 32 bit integer according to the conversion rule between bit row and integer. Functions and constants f_j, K_j, $s_1(j)$, $s_2(j)$, $l(j)$ used in each stage operation of stage i are defined as below j;

□ Boolean function and round constant

$$f_j(x, y, z) = (x \wedge y) \vee (\neg x \wedge z) \qquad (0 \le j \le 19)$$
$$f_j(x, y, z) = x \oplus y \oplus z \qquad (20 \le j \le 39,\ 60 \le j \le 79)$$
$$f_j(x, y, z) = y \oplus (x \vee \neg z) \qquad (40 \le j \le 59)$$

$$K_j = 00000000 \quad (0 \le j \le 19)$$
$$K_j = 5a827999 \quad (20 \le j \le 39) \quad \lfloor 2^{30}\sqrt{2} \rfloor$$
$$K_j = 6ed9eba1 \quad (40 \le j \le 59) \quad \lfloor 2^{30}\sqrt{3} \rfloor$$
$$K_j = 8f1bbcdc \quad (60 \le j \le 79) \quad \lfloor 2^{30}\sqrt{5} \rfloor$$

□ Circular movement amount

$$s_1(j) = 5, 11, 7, 15, 6, 13, 8, 14, 7, 12, 9, 11, 8, 15, 6, 12, 9, 14, 5, 13$$
$$(0 \le j \le 19,\ 20 \le j \le 39,\ 40 \le j \le 59,\ 60 \le j \le 79)$$

$$s_2(j) = 10 \quad (0 \le j \le 19)$$
$$s_2(j) = 17 \quad (20 \le j \le 39)$$
$$s_2(j) = 25 \quad (40 \le j \le 59)$$
$$s_2(j) = 30 \quad (60 \le j \le 79)$$

□ Message variable application order

$$l(j) = 18, 0, 1, 2, 3, 19, 4, 5, 6, 7, 16, 8, 9, 10, 11, 17, 12, 13, 14, 15 \quad (0 \le j \le 19)$$
$$l(j) = 18, 3, 6, 9, 12, 19, 15, 2, 5, 8, 16, 11, 14, 1, 4, 17, 7, 10, 13, 0 \quad (20 \le j \le 39)$$
$$l(j) = 18, 12, 5, 14, 7, 19, 0, 9, 2, 11, 16, 4, 13, 6, 15, 17, 8, 1, 10, 3 \quad (40 \le j \le 59)$$
$$l(j) = 18, 7, 2, 13, 8, 19, 3, 14, 9, 4, 16, 15, 10, 5, 0, 17, 11, 6, 1, 12 \quad (60 \le j \le 79)$$

1.2 Pseudo Code

Input : assuming that input message M was converted into $\{X_i[j]\}$ $(0 \le i < t,\ 0 \le j < 16)$, 32 bit word row with 16t through adding process.

$H_0 = 67452301;\ H_1 = \text{efcdab89};\ H_2 = \text{98badcfe};\ H_3 = 10325476;\ H_4 = \text{c3d2e1f0};$

for i = 0 to t-1 {

$A = H_0;\ B = H_1;\ C = H_2;\ D = H_3;\ E = H_4;$

for r = 0 to 3 {

j = 20 · r ;

$$X_i[16] = X_i[l(j+1)] \oplus X_i[l(j+2)] \oplus X_i[l(j+3)] \oplus X_i[l(j+4)];$$
$$X_i[17] = X_i[l(j+6)] \oplus X_i[l(j+7)] \oplus X_i[l(j+8)] \oplus X_i[l(j+9)];$$
$$X_i[18] = X_i[l(j+11)] \oplus X_i[l(j+12)] \oplus X_i[l(j+13)] \oplus X_i[l(j+14)];$$
$$X_i[19] = X_i[l(j+16)] \oplus X_i[l(j+17)] \oplus X_i[l(j+18)] \oplus X_i[l(j+19)];$$

for k = 0 to 19 {

j = 20 · r + k ;

T = $A^{<<s_1(j)} + f_j(B, C, D) + E + X_i[l(j)] + K_j;$

E = D; D = C; C = $B^{<<s_2(j)}$; B = A; A = T;

}

}

$H_0 += A;\ H_1 += B;\ H_2 += C;\ H_3 += D;\ H_4 += E;$

}

Output : output after converting 32 bit word row, H_0, H_1, H_2, H_3, H_4 into bit row in each turn

Annex 2. Test values

Annex 2.1 reference values

hash("")
= 30 79 64 ef 34 15 1d 37 c8 04 7a de c7 ab 50 f4 ff 89 76 2d
hash("a")
= 48 72 bc bc 4c d0 f0 a9 dc 7c 2f 70 45 e5 b4 3b 6c 83 0d b8
hash("abc")
= 97 5e 81 04 88 cf 2a 3d 49 83 84 78 12 4a fc e4 b1 c7 88 04
hash("message digest")
= 23 38 db c8 63 8d 31 22 5f 73 08 62 46 ba 52 9f 96 71 0b c6
hash("abcdefghijklmnopqrstuvwxyz")
= 59 61 85 c9 ab 67 03 d0 d0 db b9 87 02 bc 0f 57 29 cd 1d 3c
hash("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789")
= cb 5d 7e fb ca 2f 02 e0 fb 71 67 ca bb 12 3a f5 79 57 64 e5
hash("12345678901234567890123456789012345678901234567890123456789012345678901234567890")
= 07 f0 5c 8c 07 73 c5 5c a3 a5 a6 95 ce 6a ca 4c 43 89 11 b5
hash(a million of "a")
= d6 ad 6f 06 08 b8 78 da 9b 87 99 9c 25 25 cc 84 f4 c9 f1 8d

Annex 2.2 detailed example 1

M = "abc" = 61 62 63

X[0] = 80636261
X[1] = 00000000
X[2] = 00000000
X[3] = 00000000
X[4] = 00000000
X[5] = 00000000
X[6] = 00000000
X[7] = 00000000
X[8] = 00000000
X[9] = 00000000
X[10] = 00000000
X[11] = 00000000
X[12] = 00000000
X[13] = 00000000
X[14] = 00000018
X[15] = 00000000

Round 1 :: X[16] = 80636261
Round 1 :: X[17] = 00000000
Round 1 :: X[18] = 00000000
Round 1 :: X[19] = 00000018

Round 2 :: X[16] = 00000000
Round 2 :: X[17] = 00000000
Round 2 :: X[18] = 00000018
Round 2 :: X[19] = 80636261

Round 3 :: X[16] = 00000018
Round 3 :: X[17] = 80636261
Round 3 :: X[18] = 00000000
Round 3 :: X[19] = 00000000

Round 4 :: X[16] = 00000000
Round 4 :: X[17] = 00000018
Round 4 :: X[18] = 80636261
Round 4 :: X[19] = 00000000

H0 = 67452301

H1 = efcdab89

H2 = 98badcfe

H3 = 10325476

H4 = c3d2e1f0

j		A	B	C	D	E
j = 0	::	45321f1a	67452301	36ae27bf	98badcfe	10325476
j = 1	::	e04d88ff	45321f1a	148c059d	36ae27bf	98badcfe
j = 2	::	f60b82ab	e04d88ff	c87c6914	148c059d	36ae27bf
j = 3	::	ccd02fd8	f60b82ab	3623ff81	c87c6914	148c059d
j = 4	::	870fe765	ccd02fd8	2e0aafd8	3623ff81	c87c6914
j = 5	::	038d19e6	870fe765	40bf6333	2e0aafd8	3623ff81
j = 6	::	eb4d513d	038d19e6	3f9d961c	40bf6333	2e0aafd8
j = 7	::	c6199cc0	eb4d513d	3467980e	3f9d961c	40bf6333
j = 8	::	826359a2	c6199cc0	3544f7ad	3467980e	3f9d961c
j = 9	::	a99e52d0	826359a2	66730318	3544f7ad	3467980e
j = 10	::	28d842cf	a99e52d0	8d668a09	66730318	3544f7ad
j = 11	::	c6c273fb	28d842cf	794b42a6	8d668a09	66730318
j = 12	::	d655c964	c6c273fb	610b3ca3	794b42a6	8d668a09
j = 13	::	eb2425da	d655c964	09cfef1b	610b3ca3	794b42a6
j = 14	::	63a4b6e3	eb2425da	57259359	09cfef1b	610b3ca3
j = 15	::	f0693e36	63a4b6e3	90976bac	57259359	09cfef1b
j = 16	::	f0d180b3	f0693e36	92db8d8e	90976bac	57259359
j = 17	::	4831dd1b	f0d180b3	a4f8dbc1	92db8d8e	90976bac
j = 18	::	39ad9cba	4831dd1b	4602cfc3	a4f8dbc1	92db8d8e
j = 19	::	2b3ba486	39ad9cba	c7746d20	4602cfc3	a4f8dbc1
j = 20	::	1fcb2490	2b3ba486	3974735b	c7746d20	4602cfc3
j = 21	::	cee58557	1fcb2490	490c5677	3974735b	c7746d20
j = 22	::	046c945c	cee58557	49203f96	490c5677	3974735b
j = 23	::	aceedbe0	046c945c	0aaf9dcb	49203f96	490c5677
j = 24	::	2728fe3c	aceedbe0	28b808d9	0aaf9dcb	49203f96
j = 25	::	d2c6ef67	2728fe3c	b7c159dd	28b808d9	0aaf9dcb
j = 26	::	e4732e6e	d2c6ef67	fc784e51	b7c159dd	28b808d9
j = 27	::	e8563479	e4732e6e	decfa58d	fc784e51	b7c159dd
j = 28	::	0422d61c	e8563479	5cddc8e6	decfa58d	fc784e51
j = 29	::	eea0e13e	0422d61c	68f3d0ac	5cddc8e6	decfa58d
j = 30	::	ab216b59	eea0e13e	ac380845	68f3d0ac	5cddc8e6
j = 31	::	ed2649af	ab216b59	c27ddd41	ac380845	68f3d0ac
j = 32	::	af24b8a7	ed2649af	d6b35642	c27ddd41	ac380845
j = 33	::	5cf71c1c	af24b8a7	935fda4c	d6b35642	c27ddd41
j = 34	::	458f929a	5cf71c1c	714f5e49	935fda4c	d6b35642
j = 35	::	e9470c4c	458f929a	3838b9ee	714f5e49	935fda4c
j = 36	::	88f362f4	e9470c4c	25348b1f	3838b9ee	714f5e49

j = 37 :: 98da38db 88f362f4 1899d28e 25348b1f 3838b9ee
j = 38 :: 63608a5f 98da38db c5e911e6 1899d28e 25348b1f
j = 39 :: 57114f38 63608a5f 71b731b4 c5e911e6 1899d28e
j = 40 :: 745f8524 57114f38 bec6c114 71b731b4 c5e911e6
j = 41 :: 928b2f98 745f8524 70ae229e bec6c114 71b731b4
j = 42 :: 2bfa870f 928b2f98 48e8bf0a 70ae229e bec6c114
j = 43 :: 485b83bd 2bfa870f 3125165f 48e8bf0a 70ae229e
j = 44 :: 8543cf31 485b83bd 1e57f50e 3125165f 48e8bf0a
j = 45 :: 0234fa06 8543cf31 7a90b707 1e57f50e 3125165f
j = 46 :: f4d7e359 0234fa06 630a879e 7a90b707 1e57f50e
j = 47 :: 6a7ddb44 f4d7e359 0c0469f4 630a879e 7a90b707
j = 48 :: 194bd76a 6a7ddb44 b3e9afc6 0c0469f4 630a879e
j = 49 :: d771855c 194bd76a 88d4fbb6 b3e9afc6 0c0469f4
j = 50 :: 33743c28 d771855c d43297ae 88d4fbb6 b3e9afc6
j = 51 :: e7edeff5 33743c28 b9aee30a d43297ae 88d4fbb6
j = 52 :: 67f27cb1 e7edeff5 5066e878 b9aee30a d43297ae
j = 53 :: 39004ed5 67f27cb1 ebcfdbdf 5066e878 b9aee30a
j = 54 :: 6cd12861 39004ed5 62cfe4f9 ebcfdbdf 5066e878
j = 55 :: b229d753 6cd12861 aa72009d 62cfe4f9 ebcfdbdf
j = 56 :: 05dbaade b229d753 c2d9a250 aa72009d 62cfe4f9
j = 57 :: f1d5af33 05dbaade a76453ae c2d9a250 aa72009d
j = 58 :: ee9d7f0d f1d5af33 bc0bb755 a76453ae c2d9a250
j = 59 :: 276963ea ee9d7f0d 67e3ab5e bc0bb755 a76453ae
j = 60 :: d9855335 276963ea 7ba75fc3 67e3ab5e bc0bb755
j = 61 :: b0eeba74 d9855335 89da58fa 7ba75fc3 67e3ab5e
j = 62 :: 9a54f69e b0eeba74 766154cd 89da58fa 7ba75fc3
j = 63 :: d568200c 9a54f69e 2c3bae9d 766154cd 89da58fa
j = 64 :: 330c25d9 d568200c a6953da7 2c3bae9d 766154cd
j = 65 :: e9feeb40 330c25d9 355a0803 a6953da7 2c3bae9d
j = 66 :: 5b05bcdf e9feeb40 4cc30976 355a0803 a6953da7
j = 67 :: 3550bb91 5b05bcdf 3a7fbad0 4cc30976 355a0803
j = 68 :: 9a8c9cf2 3550bb91 d6c16f37 3a7fbad0 4cc30976
j = 69 :: 7f9c5e70 9a8c9cf2 4d542ee4 d6c16f37 3a7fbad0
j = 70 :: 037235cc 7f9c5e70 a6a3273c 4d542ee4 d6c16f37
j = 71 :: 8bf6e3d6 037235cc 1fe7179c a6a3273c 4d542ee4
j = 72 :: 8d89c7b7 8bf6e3d6 00dc8d73 1fe7179c a6a3273c
j = 73 :: ae682415 8d89c7b7 a2fdb8f5 00dc8d73 1fe7179c
j = 74 :: f9182e75 ae682415 e36271ed a2fdb8f5 00dc8d73
j = 75 :: 02d79705 f9182e75 6b9a0905 e36271ed a2fdb8f5
j = 76 :: 5327d673 02d79705 7e460b9d 6b9a0905 e36271ed
j = 77 :: 7f26992f 5327d673 40b5e5c1 7e460b9d 6b9a0905
j = 78 :: 4d5d23ff 7f26992f d4c9f59c 40b5e5c1 7e460b9d
j = 79 :: 9d3c3b96 4d5d23ff dfc9a64b d4c9f59c 40b5e5c1

H0 = 67452301 + 9d3c3b96 = 04815e97

H1 = efcdab89 + 4d5d23ff = 3d2acf88

H2 = 98badcfe + dfc9a64b = 78848349

H3 = 10325476 + d4c9f59c = e4fc4a12

H4 = c3d2e1f0 + 40b5e5c1 = 0488c7b1

HASH code = 97 5e 81 04 88 cf 2a 3d 49 83 84 78 12 4a fc e4 b1 c7 88 04

Annex 2.3 detailed example 2

```
M ="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789" =
        41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50
        51 52 53 54 55 56 57 58 59 5a 61 62 63 64 65 66
        67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76
        77 78 79 7a 30 31 32 33 34 35 36 37 38 39

First 512-bit block =
        41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50
        51 52 53 54 55 56 57 58 59 5a 61 62 63 64 65 66
        67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76
        77 78 79 7a 30 31 32 33 34 35 36 37 38 39 80 00

X[ 0] = 44434241
X[ 1] = 48474645
X[ 2] = 4c4b4a49
X[ 3] = 504f4e4d
X[ 4] = 54535251
X[ 5] = 58575655
X[ 6] = 62615a59
X[ 7] = 66656463
X[ 8] = 6a696867
X[ 9] = 6e6d6c6b
X[10] = 7271706f
X[11] = 76757473
X[12] = 7a797877
X[13] = 33323130
X[14] = 37363534
X[15] = 00803938


Round 1 :: X[16] = 10000000
Round 1 :: X[17] = 08003a3e
Round 1 :: X[18] = 00000010
Round 1 :: X[19] = 7efd454b

Round 2 :: X[16] = 263a0008
Round 2 :: X[17] = 7ef54d43
Round 2 :: X[18] = 5d575553
Round 2 :: X[19] = 6365677d

Round 3 :: X[16] = 737d7f75
```

Round 3 :: X[17] = 10101010
Round 3 :: X[18] = 05800000
Round 3 :: X[19] = 00101000

Round 4 :: X[16] = 7375777d
Round 4 :: X[17] = 5d474543
Round 4 :: X[18] = 6ee55d43
Round 4 :: X[19] = 262a1018

H0 = 67452301
H1 = efcdab89
H2 = 98badcfe
H3 = 10325476
H4 = c3d2e1f0

	A	B	C	D	E
j = 0 ::	45321f2a	67452301	36ae27bf	98badcfe	10325476
j = 1 ::	a42de8df	45321f2a	148c059d	36ae27bf	98badcfe
j = 2 ::	2e82b8b2	a42de8df	c87ca914	148c059d	36ae27bf
j = 3 ::	6fff365d	2e82b8b2	b7a37e90	c87ca914	148c059d
j = 4 ::	4ba724d9	6fff365d	0ae2c8ba	b7a37e90	c87ca914
j = 5 ::	c6f7606b	4ba724d9	fcd975bf	0ae2c8ba	b7a37e90
j = 6 ::	4c192962	c6f7606b	9c93652e	fcd975bf	0ae2c8ba
j = 7 ::	6a2e27d3	4c192962	dd81af1b	9c93652e	fcd975bf
j = 8 ::	52d226db	6a2e27d3	64a58930	dd81af1b	9c93652e
j = 9 ::	1b0c07d6	52d226db	b89f4da8	64a58930	dd81af1b
j = 10 ::	3a48e8f9	1b0c07d6	489b6d4b	b89f4da8	64a58930
j = 11 ::	bef208d3	3a48e8f9	301f586c	489b6d4b	b89f4da8
j = 12 ::	89b0db3b	bef208d3	23a3e4e9	301f586c	489b6d4b
j = 13 ::	4b59f37f	89b0db3b	c8234efb	23a3e4e9	301f586c
j = 14 ::	27351bac	4b59f37f	c36cee26	c8234efb	23a3e4e9
j = 15 ::	40c9d040	27351bac	67cdfd2d	c36cee26	c8234efb
j = 16 ::	bd8b4521	40c9d040	d46eb09c	67cdfd2d	c36cee26
j = 17 ::	2f344be5	bd8b4521	27410103	d46eb09c	67cdfd2d
j = 18 ::	eaf360a3	2f344be5	2d1486f6	27410103	d46eb09c
j = 19 ::	6e586a18	eaf360a3	d12f94bc	2d1486f6	27410103
j = 20 ::	c0f085e5	6e586a18	c147d5e6	d12f94bc	2d1486f6
j = 21 ::	da45a825	c0f085e5	d430dcb0	c147d5e6	d12f94bc
j = 22 ::	866f084e	da45a825	0bcb81e1	d430dcb0	c147d5e6
j = 23 ::	141df495	866f084e	504bb48b	0bcb81e1	d430dcb0
j = 24 ::	8e993129	141df495	109d0cde	504bb48b	0bcb81e1
j = 25 ::	44a3e18a	8e993129	e92a283b	109d0cde	504bb48b
j = 26 ::	c65e076c	44a3e18a	62531d32	e92a283b	109d0cde

j = 27 :: 0920d6da c65e076c c3148947 62531d32 e92a283b
j = 28 :: 9388f846 0920d6da 0ed98cbc c3148947 62531d32
j = 29 :: 7bb13b8b 9388f846 adb41241 0ed98cbc c3148947
j = 30 :: d72d809a 7bb13b8b f08d2711 adb41241 0ed98cbc
j = 31 :: 725e605c d72d809a 7716f762 f08d2711 adb41241
j = 32 :: ee836e69 725e605c 0135ae5b 7716f762 f08d2711
j = 33 :: 4f091795 ee836e69 c0b8e4bc 0135ae5b 7716f762
j = 34 :: 1740cd2d 4f091795 dcd3dd06 c0b8e4bc 0135ae5b
j = 35 :: 3ae274da 1740cd2d 2f2a9e12 dcd3dd06 c0b8e4bc
j = 36 :: 2b440566 3ae274da 9a5a2e81 2f2a9e12 dcd3dd06
j = 37 :: 3ab41628 2b440566 e9b475c4 9a5a2e81 2f2a9e12
j = 38 :: 6c0c6c05 3ab41628 0acc5688 e9b475c4 9a5a2e81
j = 39 :: a06ccd40 6c0c6c05 2c507568 0acc5688 e9b475c4
j = 40 :: 3d17a198 a06ccd40 0ad818d8 2c507568 0acc5688
j = 41 :: aa645397 3d17a198 8140d99a 0ad818d8 2c507568
j = 42 :: a222c158 aa645397 307a2f43 8140d99a 0ad818d8
j = 43 :: e019e372 a222c158 2f54c8a7 307a2f43 8140d99a
j = 44 :: 1dec1fb1 e019e372 b1444582 2f54c8a7 307a2f43
j = 45 :: 655a0199 1dec1fb1 e5c033c6 b1444582 2f54c8a7
j = 46 :: f6b31c29 655a0199 623bd83f e5c033c6 b1444582
j = 47 :: 72da30c0 f6b31c29 32cab403 623bd83f e5c033c6
j = 48 :: db3b55d3 72da30c0 53ed6638 32cab403 623bd83f
j = 49 :: a8fa93ca db3b55d3 80e5b461 53ed6638 32cab403
j = 50 :: 8a281e20 a8fa93ca a7b676ab 80e5b461 53ed6638
j = 51 :: b05855f0 8a281e20 9551f527 a7b676ab 80e5b461
j = 52 :: ca802c35 b05855f0 4114503c 9551f527 a7b676ab
j = 53 :: 4af6b1a9 ca802c35 e160b0ab 4114503c 9551f527
j = 54 :: e1e3a3ae 4af6b1a9 6b950058 e160b0ab 4114503c
j = 55 :: 2fa439b0 e1e3a3ae 5295ed63 6b950058 e160b0ab
j = 56 :: aa9577de 2fa439b0 5dc3c747 5295ed63 6b950058
j = 57 :: 72dbd9de aa9577de 605f4873 5dc3c747 5295ed63
j = 58 :: 5a3ebcce 72dbd9de bd552aef 605f4873 5dc3c747
j = 59 :: 3735a1ad 5a3ebcce bce5b7b3 bd552aef 605f4873
j = 60 :: a0a2b9ca 3735a1ad 968faf33 bce5b7b3 bd552aef
j = 61 :: e6045a60 a0a2b9ca 4dcd686b 968faf33 bce5b7b3
j = 62 :: 165a6ddd e6045a60 a828ae72 4dcd686b 968faf33
j = 63 :: 93adc4e5 165a6ddd 39811698 a828ae72 4dcd686b
j = 64 :: bab79c49 93adc4e5 45969b77 39811698 a828ae72
j = 65 :: 40b1fbc6 bab79c49 64eb7139 45969b77 39811698
j = 66 :: 66b25e08 40b1fbc6 6eade712 64eb7139 45969b77
j = 67 :: ee621520 66b25e08 902c7ef1 6eade712 64eb7139
j = 68 :: 2bb2f2e2 ee621520 19ac9782 902c7ef1 6eade712
j = 69 :: e92e154d 2bb2f2e2 3b988548 19ac9782 902c7ef1

```
j = 70 :: f86f2f44 e92e154d 8aecbcb8 3b988548 19ac9782
j = 71 :: 7b1ce216 f86f2f44 7a4b8553 8aecbcb8 3b988548
j = 72 :: 62cfdfbd 7b1ce216 3e1bcbd1 7a4b8553 8aecbcb8
j = 73 :: a18b2de4 62cfdfbd 9ec73885 3e1bcbd1 7a4b8553
j = 74 :: 72892a81 a18b2de4 58b3f7ef 9ec73885 3e1bcbd1
j = 75 :: 2526c7a6 72892a81 2862cb79 58b3f7ef 9ec73885
j = 76 :: f43fcc35 2526c7a6 5ca24aa0 2862cb79 58b3f7ef
j = 77 :: 8f24d2b2 f43fcc35 8949b1e9 5ca24aa0 2862cb79
j = 78 :: 06345c67 8f24d2b2 7d0ff30d 8949b1e9 5ca24aa0
j = 79 :: 6d26f10f 06345c67 a3c934ac 7d0ff30d 8949b1e9

H0 = 67452301 + 6d26f10f = d46c1410
H1 = efcdab89 + 06345c67 = f60207f0
H2 = 98badcfe + a3c934ac = 3c8411aa
H3 = 10325476 + 7d0ff30d = 8d424783
H4 = c3d2e1f0 + 8949b1e9 = 4d1c93d9


Second 512-bit Block =
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 f0 01 00 00 00 00 00 00


X[ 0] = 00000000
X[ 1] = 00000000
X[ 2] = 00000000
X[ 3] = 00000000
X[ 4] = 00000000
X[ 5] = 00000000
X[ 6] = 00000000
X[ 7] = 00000000
X[ 8] = 00000000
X[ 9] = 00000000
X[10] = 00000000
X[11] = 00000000
X[12] = 00000000
X[13] = 00000000
X[14] = 000001f0
X[15] = 00000000
```

Round 1 :: X[16] = 00000000
Round 1 :: X[17] = 00000000
Round 1 :: X[18] = 00000000
Round 1 :: X[19] = 000001f0

Round 2 :: X[16] = 00000000
Round 2 :: X[17] = 00000000
Round 2 :: X[18] = 000001f0
Round 2 :: X[19] = 00000000

Round 3 :: X[16] = 000001f0
Round 3 :: X[17] = 00000000
Round 3 :: X[18] = 00000000
Round 3 :: X[19] = 00000000

Round 4 :: X[16] = 00000000
Round 4 :: X[17] = 000001f0
Round 4 :: X[18] = 00000000
Round 4 :: X[19] = 00000000

H0 = d46c1410
H1 = f60207f0
H2 = 3c8411aa
H3 = 8d424783
H4 = 4d1c93d9

	A	B	C	D	E
j = 0 ::	17df5796	d46c1410	081fc3d8	3c8411aa	8d424783
j = 1 ::	b08af9fb	17df5796	b0504351	081fc3d8	3c8411aa
j = 2 ::	9a51d2da	b08af9fb	7d5e585f	b0504351	081fc3d8
j = 3 ::	21e76b5b	9a51d2da	2be7eec2	7d5e585f	b0504351
j = 4 ::	997ae4e0	21e76b5b	474b6a69	2be7eec2	7d5e585f
j = 5 ::	e53e5c47	997ae4e0	9dad6c87	474b6a69	2be7eec2
j = 6 ::	496da530	e53e5c47	eb938265	9dad6c87	474b6a69
j = 7 ::	aa2a9d89	496da530	f9711f94	eb938265	9dad6c87
j = 8 ::	9eef38b1	aa2a9d89	b694c125	f9711f94	eb938265
j = 9 ::	d2701f68	9eef38b1	aa7626a8	b694c125	f9711f94
j = 10 ::	8426d2dc	d2701f68	bce2c67b	aa7626a8	b694c125
j = 11 ::	a591cc2e	8426d2dc	c07da349	bce2c67b	aa7626a8
j = 12 ::	f526dbb8	a591cc2e	9b4b7210	c07da349	bce2c67b
j = 13 ::	ec2ca44f	f526dbb8	4730ba96	9b4b7210	c07da349
j = 14 ::	1b1071d4	ec2ca44f	9b6ee3d4	4730ba96	9b4b7210
j = 15 ::	2da56e95	1b1071d4	b2913fb0	9b6ee3d4	4730ba96
j = 16 ::	248c9881	2da56e95	41c7506c	b2913fb0	9b6ee3d4

j = 17 :: 55247e1b 248c9881 95ba54b6 41c7506c b2913fb0

j = 18 :: 9cec55f6 55247e1b 32620492 95ba54b6 41c7506c

j = 19 :: 5d4028bf 9cec55f6 91f86d54 32620492 95ba54b6

j = 20 :: d7b8245a 5d4028bf abed39d8 91f86d54 32620492

j = 21 :: b55cd11b d7b8245a 517eba80 abed39d8 91f86d54

j = 22 :: c80f1bc9 b55cd11b 48b5af70 517eba80 abed39d8

j = 23 :: 40ec5c63 c80f1bc9 a2376ab9 48b5af70 517eba80

j = 24 :: 09a62ae9 40ec5c63 3793901e a2376ab9 48b5af70

j = 25 :: 3dddf101 09a62ae9 b8c681d8 3793901e a2376ab9

j = 26 :: 619e20be 3dddf101 55d2134c b8c681d8 3793901e

j = 27 :: eb0f05b3 619e20be e2027bbb 55d2134c b8c681d8

j = 28 :: 711a1daf eb0f05b3 417cc33c e2027bbb 55d2134c

j = 29 :: 9aa1412a 711a1daf 0b67d61e 417cc33c e2027bbb

j = 30 :: ba085316 9aa1412a 3b5ee234 0b67d61e 417cc33c

j = 31 :: 893067a5 ba085316 82553542 3b5ee234 0b67d61e

j = 32 :: 99557b90 893067a5 a62d7410 82553542 3b5ee234

j = 33 :: 00f1cf6e 99557b90 cf4b1260 a62d7410 82553542

j = 34 :: 097ea83b 00f1cf6e f72132aa cf4b1260 a62d7410

j = 35 :: 23cf8de4 097ea83b 9edc01e3 f72132aa cf4b1260

j = 36 :: 296cefb2 23cf8de4 507612fd 9edc01e3 f72132aa

j = 37 :: 7af5d598 296cefb2 1bc8479f 507612fd 9edc01e3

j = 38 :: baebe95b 7af5d598 df6452d9 1bc8479f 507612fd

j = 39 :: e67dc4d1 baebe95b ab30f5eb df6452d9 1bc8479f

j = 40 :: 6c25e610 e67dc4d1 b775d7d2 ab30f5eb df6452d9

j = 41 :: bef8dae2 6c25e610 a3ccfb89 b775d7d2 ab30f5eb

j = 42 :: 65db689f bef8dae2 20d84bcc a3ccfb89 b775d7d2

j = 43 :: b8c30d8a 65db689f c57df1b5 20d84bcc a3ccfb89

j = 44 :: 7dec56e2 b8c30d8a 3ecbb6d1 c57df1b5 20d84bcc

j = 45 :: 9e974045 7dec56e2 1571861b 3ecbb6d1 c57df1b5

j = 46 :: b425fce9 9e974045 c4fbd8ad 1571861b 3ecbb6d1

j = 47 :: 6744b0c3 b425fce9 8b3d2e80 c4fbd8ad 1571861b

j = 48 :: 5abca4ea 6744b0c3 d3684bf9 8b3d2e80 c4fbd8ad

j = 49 :: a2d323ff 5abca4ea 86ce8961 d3684bf9 8b3d2e80

j = 50 :: 98d058e5 a2d323ff d4b57949 86ce8961 d3684bf9

j = 51 :: f44f7316 98d058e5 ff45a647 d4b57949 86ce8961

j = 52 :: 89bb04a6 f44f7316 cb31a0b1 ff45a647 d4b57949

j = 53 :: 05b184d6 89bb04a6 2de89ee6 cb31a0b1 ff45a647

j = 54 :: 6a988871 05b184d6 4d137609 2de89ee6 cb31a0b1

j = 55 :: 5d3736d1 6a988871 ac0b6309 4d137609 2de89ee6

j = 56 :: 6228183f 5d3736d1 e2d53110 ac0b6309 4d137609

j = 57 :: 7f1fca1b 6228183f a2ba6e6d e2d53110 ac0b6309

j = 58 :: dc6f42ab 7f1fca1b 7ec45030 a2ba6e6d e2d53110

j = 59 :: 3ba023e9 dc6f42ab 36fe3f94 7ec45030 a2ba6e6d

j = 60 :: 3a2fd57f 3ba023e9 f71bd0aa 36fe3f94 7ec45030
j = 61 :: 86d1d3b4 3a2fd57f 4ee808fa f71bd0aa 36fe3f94
j = 62 :: b2dfe3e2 86d1d3b4 ce8bf55f 4ee808fa f71bd0aa
j = 63 :: 7edb1506 b2dfe3e2 21b474ed ce8bf55f 4ee808fa
j = 64 :: f2a969c5 7edb1506 acb7f8f8 21b474ed ce8bf55f
j = 65 :: 7eb909a3 f2a969c5 9fb6c541 acb7f8f8 21b474ed
j = 66 :: 2b8229c3 7eb909a3 7caa5a71 9fb6c541 acb7f8f8
j = 67 :: 63ea1937 2b8229c3 dfae4268 7caa5a71 9fb6c541
j = 68 :: ac654fa8 63ea1937 cae08a70 dfae4268 7caa5a71
j = 69 :: d7657342 ac654fa8 d8fa864d cae08a70 dfae4268
j = 70 :: f82fc887 d7657342 2b1953ea d8fa864d cae08a70
j = 71 :: fcc72df2 f82fc887 b5d95cd0 2b1953ea d8fa864d
j = 72 :: 9633fde2 fcc72df2 fe0bf221 b5d95cd0 2b1953ea
j = 73 :: 703bdee2 9633fde2 bf31cb7c fe0bf221 b5d95cd0
j = 74 :: 2af69707 703bdee2 a58cff78 bf31cb7c fe0bf221
j = 75 :: 611f0e82 2af69707 9c0ef7b8 a58cff78 bf31cb7c
j = 76 :: 9fdf2ce1 611f0e82 cabda5c1 9c0ef7b8 a58cff78
j = 77 :: 378d8146 9fdf2ce1 9847c3a0 cabda5c1 9c0ef7b8
j = 78 :: ea0027da 378d8146 67f7cb38 9847c3a0 cabda5c1
j = 79 :: 271249bb ea0027da 8de36051 67f7cb38 9847c3a0

H0 = 67452301 + 271249bb = fb7e5dcb
H1 = efcdab89 + ea0027da = e0022fca
H2 = 98badcfe + 8de36051 = ca6771fb
H3 = 10325476 + 67f7cb38 = f53a12bb
H4 = c3d2e1f0 + 9847c3a0 = e5645779

HASH code = cb 5d 7e fb ca 2f 02 e0 fb 71 67 ca bb 12 3a f5 79 57 64 e5

Contributors to Standard Write-up

Standard No. : TTAS.KO-12.0011/R1

The following individuals have contributed to enacting, amending, publishing of the present standard.

Task	Name	Committee & Position	Contact	Company
Assignment Proposal		Cryptographic Technology Team(SG10.02)		KISA
First Standard Draft Submission	S.J. Kim	Cryptographic Technology Team /Chairman	(02)3488-4066	KISA
First Standard Draft Review & Write-up	S.J. Kim	Cryptographic Technology Team /Chairman	(02)3488-4066	KISA
	S.J. Park	Cryptographic Technology Team /Vice chairman	(02)3453-1114	BCQRE
	S.J. Lee	Cryptographic Technology Team /Staff	(02)3488-4171	KISA
	H.J. Gwon	Cryptographic Technology Team /Committeeman	(02)3488-4271	KISA
	I.S. Lee	Cryptographic Technology Team /Committeeman	(02)3453-1114	BCQRE
	S.W. Sin	Cryptographic Technology Team /Observer	(042)860-5820	KISA
	C.H. Im	Cryptographic Technology Team /Observer	(02)578-8925	Future System
Standard Editing & Supervision	S.J. Kim	Cryptographic Technology Team /Chairman	(02)3488-4066	KISA
	S.J. Park	Cryptographic Technology Team /Vice chairman	(02)3453-1114	BCQRE
	S.J. Lee	Cryptographic Technology Team /Staff	(02)3488-4171	KISA
	H.J. Gwon	Cryptographic Technology Team /Committeeman	(02)3488-4271	KISA
	S.W. Sin	Cryptographic Technology Team /Observer	(042)860-5820	ETRI
	C.H. Im	Cryptographic Technology Team /Observer	(02)578-8925	Future System
	J.H. No	Cryptographic Technology Team /Observer	(031)450-7885	LG Information & Communications Co., Ltd

Standard Deliberation	H.S. Lee	Cryptographic Technology Team /Chairman	(02)3488-4150	KISA
	S.W. Kim	Cryptographic Technology Team /Vice chairman	(031)450-5025	Hansei Univ.
	H.B. Kim	Cryptographic Technology Team /Committeeman	(02)3488-4213	KISA
	S.J. Kim	Cryptographic Technology Team /Committeeman	(02)3488-4066	KISA
	I.S. Leek	Cryptographic Technology Team /Committeeman	(042)870-8047	KT
	J.O. Sin	Cryptographic Technology Team /Special committeeman	(02)2260-3336	Dongguk Univ.
	Y.R. Choe	Cryptographic Technology Team /Special committeeman	(042)280-2541	Daejeon Univ.
	S.C. Go	Cryptographic Technology Team /Committeeman	(02)3488-4100	KISA
	K.K. Lee	Cryptographic Technology Team /Committeeman	(02)3488-4030	KISA
Bureau	M.H. Gwon	Cryptographic Technology Standard Department /chief	(02)723-7073	TTA
	M.C. Lee	Information Technology Standard Department	(02)723-7073	TTA