ELSEVIER

Information Sciences 118 (1999) 223–239

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

One-way hash functions with changeable parameters

Tsu-Miin Hsieh [a,1], Yi-Shiung Yeh [a], Chu-Hsing Lin [b,*], Ssu-Heng Tuan [a]

[a] *Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan*

[b] *Department of Computer and Information Sciences, Tunghai University, Taichung, Taiwan*

Received 1 April 1998; received in revised form 13 April 1999; accepted 23 June 1999

Abstract

In this paper, four new algorithms for improvements of MD5 and SHA-Keyed/Unkeyed MD5 and Keyed/Unkeyed SHA are proposed. We change the mode of fixed retrieval of the parameters to that of dynamic retrieval of the parameters by applying a mapping between the index phrase and the parameter table. As shown in the experimental result, with 1% degradation of performance, comparing to the original algorithms, dynamic retrieval of parameters can be achieved. We believe that the security can be increased under the improvements. © 1999 Elsevier Science Inc. All rights reserved.

Keywords: MD5; SHA; One-way hash functions; Keyed/Unkeyed MD5; Keyed/Unkeyed SHA

1. Introduction

MD5 was proposed by R. Rivest and the RSA Data Secure, in 1991, and enrolled into No. 1321 suggestion of Request for Comments (RFC) in 1992. SHA was announced by the United State Federal Register in 1992, and enrolled into No. 180 [6] suggestion of Federal Information Processing Standard (FIPS) by National Institute of Standards and Technology (NIST) in 1993.

* Corresponding author. Fax: +886-04-359-6557; e-mail: chlin@mail.thu.edu.tw
[1] E-mail: tmhsieh@csie.nctu.edu.tw

MD5 [19,4,10] and SHA [9] are two important one-way hash functions used for digital signature in cryptography. These algorithms process a variable-length message and output a fixed length of Message Digest. However, it seems to have some predefined rule for MD5 and SHA: The algorithms will use some particular parameters when some steps are executing. That is, the two algorithms have fixed and sequential retrievals of predefined parameters. It may be vulnerable to attackers. In this paper, we propose four improvements of MD5 and SHA-Keyed/Unkeyed MD5 and Keyed/Unkeyed SHA. We change the mode of fixed retrieval of the parameters to that of dynamic retrieval of the parameters by applying a mapping between the index phrase and the parameter table. As shown in the experimental result, under 1% degradation of performance, comparing to the original algorithms, dynamic retrieval of parameters can be achieved. In the remainder of this section, for better understanding the algorithms of MD5 and SHA are briefly reviewed. In Section 2, cryptanalysis on MD5 and SHA are introduced. Some existing improved algorithms for MD5 and SHA are described in Section 3. Dynamic retrieval of parameter table for one-way hash function is introduced in Section 4. Keyed/Unkeyed MD5 and Keyed/Unkeyed SHA are proposed in Section 5. Finally, we have the experimental result in Section 7.

1.1. MD5

MD5 [19] is an algorithm designed for 32-bit machines, it can convert a message with arbitrary length to a fixed length of 128-bit Message Digest. The original purpose of designing MD5 is for digital signatures. A large file can be compressed first by MD5, then encrypted by RSA-like schemes, and thus delivered to the destination. MD5 is depicted by Fig. 1.

1.2. SHA (Secure Hash Algorithm)

SHA [9] is also an algorithm designed for 32-bit machines. Evolving from MD4, some steps in SHA is resemble with MD5. Similarly, SHA can convert an arbitrary length of message to a fixed length of 160-bit Message Digests. SHA is originally applied to Digital Signature Standard (DSA) that is proposed by the NIST. SHA is depicted by Fig. 2.

2. Cryptanalysis

2.1. Attacks on MD5

Since the announcement of MD5 in 1991, some attacks have been proposed. In the following, the attacking methods are briefly described.

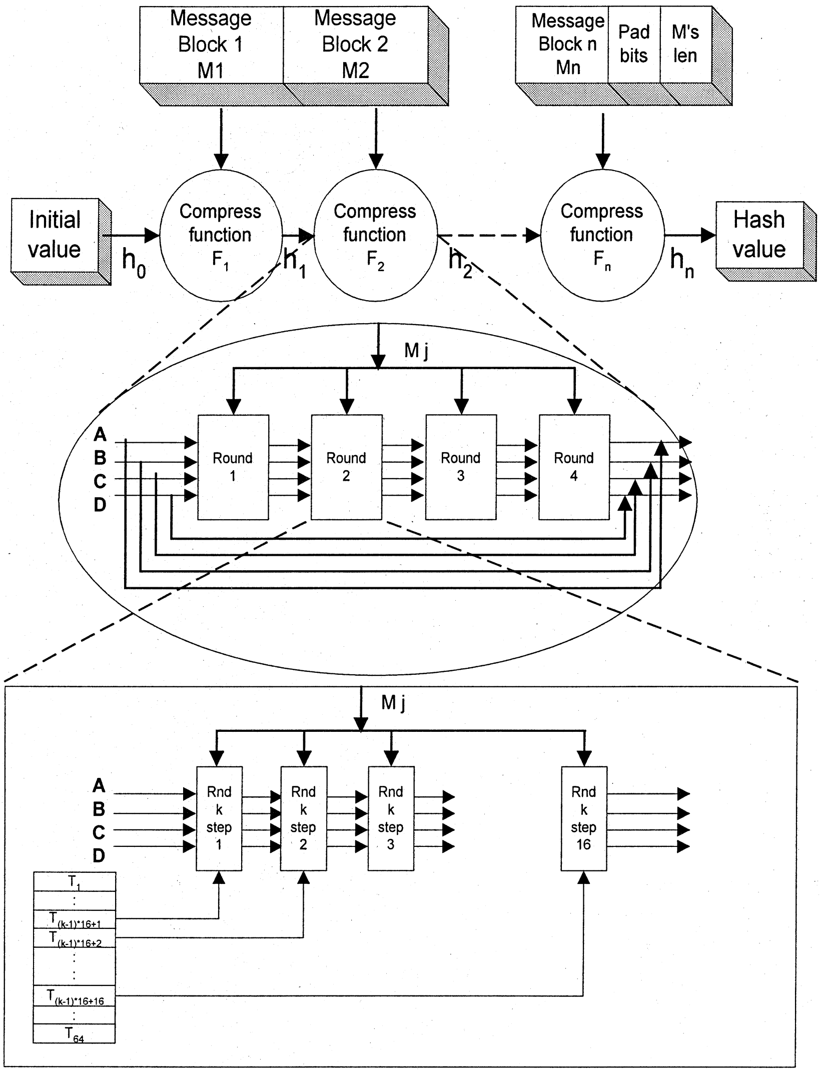

Fig. 1. MD5.

(a) *Differential cryptanalysis*: Differential attack is one type of cryptanalysis that is based on the observation of the difference of a ciphertext pair H and H' of two distinct messages M and M', respectively. It was proposed by Biham and Shamir in 1990 [1] for the attacks on DES and its related encryption schemes. Afterwards, similar cryptanalytic methods are used on Feal, N-Hash, PES, Snefru, Khafre, REDOC-II, LOKI and Lucifer, etc. [2,5].

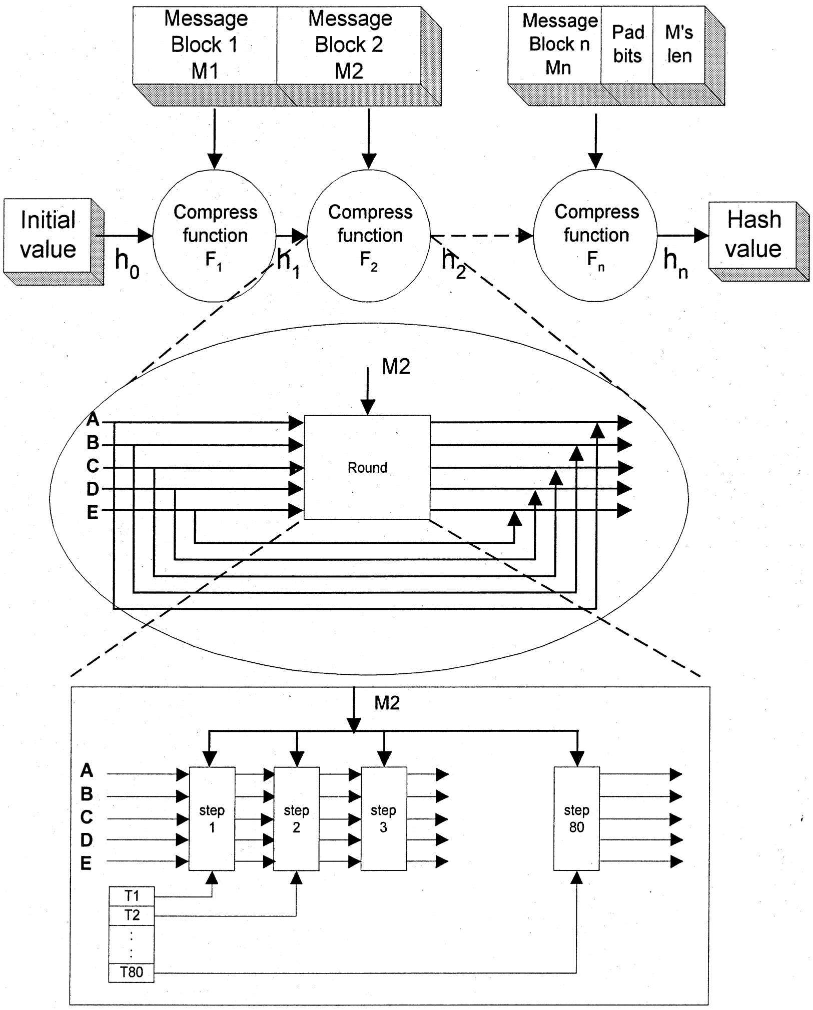

Fig. 2. SHA.

The spirit is to make use of the weakness: abuses of XOR operations. Because of an XOR operation is equal to the operation of addition mod 2. Differential method can be used to compare the difference between a plaintext and its ciphertext, and estimate the distance (mod 2) of the difference. Thus the effect of using XOR operations will be cut down. Since a lot of XOR operations are used in MD5, it becomes an object of differential cryptanalysis.

In 1992 Thomas A. Berson proposed a differential attack, based on the characteristic of weakly collision free, with modulus 2^{32} [3]. At first, the attacked message M and a forgery message M' are decided such that $H(M) = H(M')$. Then it attacks on some SubBlock_i in one time, and further on the other subblocks. However, this cryptanalysis can only handle operations in one round, not for operations in four rounds.

(b) *Pseudo-collision*: In MD5, a real collision occurs if we have two messages M_1 and M_2 such that $H(\text{IV}, M_1) = H(\text{IV}, M_2)$, where H is an one-way hash function and IV is the Initial Chaining Variables.

In 1994 Bert den Boer and Antoon Bosselaers proposed the concept of pseudo collision [8]. It means that $H(\text{IV}, M_1) = H(\text{IV}_2, M_2)$, where IV_1 and IV_2 are two initial values and differ from IV. They discover that it needs only 2^{16} operations to find the Pseudo Collision of MD5 at particular IV of initial values. However, it needs 2^{64} operations to look for the Real Collision in case of weakly collision free and 2^{128} operations in case of strong collision free. In this method the values of IV are decided by the attacker and not for general case of MD5.

(c) *Collision of the compress function in MD*5: According to the pigeon hole principle, the exhaustive approach to discover a real collision is to take $2^{128} + 1$ messages, M_k's, and compress them with MD5 algorithm to obtain 128-bits outputs, MD_k's, respectively. Then we will have at least two message namely M_i and M_j with the same output; that is: $H(\text{IV}, M_i) = H(\text{IV}, M_j)$.

In 1996 Hans Dobbertin discovered the collision, based on pseudo collision and defining IV value by himself, with a Pentium PC in 10 h [14–16]. Though this does not show that collision can be found at the circumstances of standard IV, it exposes the question: is 128-bits output enough for MD5 to be hard to reverse?

2.2. Attacks on SHA

The cryptanalysis of SHA is not yet found in thesis now.

3. The improved algorithms

There are some improvements of MD5 and SHA and can be categorized as two methods. One method is to lengthen the message digest and thus to reduce the probability of collision. For example: the variable length of output in HAVAL, the 160-bit output in RIPEMD-160 [17,20]. The other method is to enhance the characteristics of one-way hashing and thus it is hard to reverse. For example: one left-shift operation is added in SHA-1, the round number is increased in RIPEMD-160, S-boxes are added in Tiger and a key is added in HMAC-MD5 [13,18].

Note that SHA-1 is an improvement of SHA; while HAVAL, RIPEMD-160, Tiger and HMAC-MD5 are the improvements of MD5. For comparison, we will introduce these improvements briefly as follows.

3.1. SHA-1

The main idea of SHA-1 [11] is to add a shift operation at the padding of $X[16..79]$ in the old version SHA; that is, $X[j] = (X[j-3] \oplus X[j-8] \oplus X[j-14] \oplus X[j-16]) \ll 1$, and $\ll 1$ represents a left shift 1 bit.

3.2. HAVAL

HAVAL was proposed by Zheng et al. in 1993 [7] as an improvement of MD5. Message is divided into subblocks of 1024 bits instead of 512 bits. Besides, two parameters are controlled and changeable: the round number (can be three types of 3, 4 or 5 rounds) and the length of Message Digest (can be the five kinds of 128, 160, 192, 224 or 256 bits). Therefore there are fifteen kinds of possible selections for users to determine based on the consideration of different grades of security and efficiency. Note that the length of input message M should be smaller than 2^{64} bits.

3.3. RIPEMD-160

In 1996 Hans Dobbertin et al. proposed RIPEMD-160 [17,20] algorithm. It divides an input message into n subblocks of 512 bits each and outputs a message digest of length 160 bits. The authors hope that it can replace the old MD4, MD5 and RIPEMD-128 algorithms. The reason is that the output length (128 bits) of the above three algorithms is too short, a cryptanalytic method may be found very soon. They show that the RIPEMD-160 can work well within the next decade. However, RIPEMD-160 has a lower efficiency than SHA-1 for 20% in computation time. Note that the length of input message is smaller than 2^{64} bits.

3.4. Tiger

In 1996 Anderson and Biham proposed the Tiger algorithm [12] primarily for 64 bits machine (such as DEC Alpha). They hope it will be suitable for processors in the next five years. An arbitrary length of message is input and pass through three rounds of operations. Each round adopt 8-to-64 S-boxes (substitution) to increase the randomness and to prevent the differential cryptanalysis. Finally, it output a 192-bits hash value (For compatible with the application environment now, it can be taken for 160 or 128 bits). The author shows that even if the length of output taken is the same as MD5 or SHA-1, Tiger has still higher security. For efficiency by comparing to SHA-1, Tiger is quicker on 16-bits machines, about has the same speed on 32-bits machines, and quicker with a multiplicity of 2.5 on 64 machines.

3.5. HMAC-MD5 (keyed MD5)

Message Authentication Code (MAC) is a method used for authenticating a message. By applying the keys for encryption and decryption between the sender and the receiver a message can be authenticated. There are two methods for implementing MAC: (1) using block ciphers such as DES-like schemes and (2) using hashing functions such as MD5 or SHA. Because the former has problems of export constraint and less efficiency, it is convenient to use hash functions. However, MD5 and SHA are not originally designed for MAC – there is no keys. It may need some modifications to these hash functions for computing MAC. The first one is that a key is appended after data, i.e. key is seen as part of data. The second is that IV is seen as key, i.e. instead of the standard IV, the sender and the receiver use the same private key to compute the same message digest. One example of using the second method, indicated as HMAC-MD5 [18], was proposed by Krawczyk, Bellar and Canetti in 1996. The algorithm of HMAC-MD5 can be denoted as

$$\mathrm{MD5}(\mathrm{KXOR}\ 0\times 36\ldots 36, \mathrm{MD5}(\mathrm{KXOR}\ 0\times 5\mathrm{C}\ldots 5\mathrm{C}, \mathrm{text})).$$

The HMAC-MD5 does not change the algorithm of MD5, thus can retain the performance. Further, other hash functions can also be used to replace the MD5. Since a key is used as IV, there is one basic requirement in the chosen hash function – hard to find collision for any IV value. Besides, the length of key must be not less than 128 bits (the length of IV). An application program will transform a longer key to a K value of 128 bits and then uses it as IV to run HMAC-MD5.

4. Our algorithm

In this section, we will propose improvements for MD5 and SHA. Instead of fixed retrieval of T_i values from tables, changeable parameters are used in our schemes.

4.1. Fixed retrieval of parameters in MD5 and SHA

As stated previously MD5 has 68 parameters, four are the initial values A, B, C and D of the chaining variables and the other 64 are parameters of T_i values, where $\mathrm{Ti} = 2^{32} * \mathrm{abs}(\sin(i+1)), 0 \leqslant i \leqslant 63$, for i in radians. SHA have 85 parameters, five are the initial values A, B, C, D and E of the chaining variables and the other 80 are parameters of T_i values indicated as

$$\begin{aligned} T_i &= 2^{1/2}/4 = 0 \times 5a827999 \quad \text{for } 0 \leqq i \leqq 19 \\ &= 3^{1/2}/4 = 0 \times 6ed9eba1 \quad \text{for } 20 \leqq i \leqq 39 \\ &= 5^{1/2}/4 = 0 \times 8f1bbcdc \quad \text{for } 40 \leqq i \leqq 59 \\ &= 10^{1/2}/4 = 0 \times ca62c1d6 \quad \text{for } 60 \leqq i \leqq 79. \end{aligned}$$

The retrieval of the parameters is sequential and fixed in MD5 and SHA. For instance in MD5, at the beginning it needs four initial parameters, thereupon it takes A, B, C, D, and then there are 64 operation steps, it takes one by one from T_0 to T_{63}. In SHA, at the beginning it needs five parameters, thereupon it takes A, B, C, D, E, and then there are 80 operation steps, it takes one by one from T_0 to T_{79}.

We can see that there exist some predefined rules for MD5 and SHA: it will use some particular parameters when some steps of the algorithm are executing. Though we do not know if this is the entry point of attackers, we can seal-off this entry point. We think that this is a place worthy of improving – change the mode of fixed retrieval of the parameters to that of dynamic retrieval of the parameters.

4.2. Dynamic retrieval of parameters with a mapping

There are lots of methods that can be applied for dynamic retrieval of parameters. For instance, we may make use of a mapping table: Deposit the parameters in a table T in advance and use a mapping function to retrieve them in some order when needed. The advantage of using a mapping table is that it can prevent to take repeated parameters. For this reason, to avoid the duplicate retrievals, the number of parameters in the table must be more than that is required for MD5 and SHA. Therefore, assume that the size of the table T is n, n not smaller than 68 (for MD5) or 85 (for SHA), the permutations of arbitrarily choose from n are $P(n, 68)$ or $P(n, 85)$. Compare with the original MD5 and SHA which have only one type of sequence, our method is dynamic and would be more secure.

However, the problem is: How to design the above mapping table? An example is to use a "hashing" that is studied in the course of Data Structure. There are several methods for implementing hashing: The square and median number method, the prime number division, the fold method, and the digit analysis method, etc. For more detailed please cite any textbook in Data Structure.

Regarding MD5 as an example, we may pick up any 68 parameters from the table of size n. When $n = 256$, we can use 8 bits to indicate the index values. To attain the purpose, we only need to find 68 bytes of index values from the table. Similarly, for SHA, 85 bytes of index values are required from the table of size 256.

For convenience, we concatenate the series of index values and indicate as the Index Phrase $\boldsymbol{I}, \boldsymbol{I} = \{I_1, I_2, \ldots, I_k\},\ k = 68$ (for MD5) or $k = 85$ (for SHA). The value of I_j in $\boldsymbol{I}, (1 \leqq j \leqq k)$, means that it will retrieve the I_jth item in table $\boldsymbol{T}$, i.e., $\boldsymbol{T}[I_j]$. For example: let $\boldsymbol{I} = \{163, 22, 97, 48, \ldots\}$, the 3rd item indicates to retrieve the 97th parameter, $\boldsymbol{T}97$, from the table.

In the proposed schemes, to prevent to fall into the style of fixed retrievals of parameters, the index phrases should be changeable as needed. We consider two methods to attain this purpose:

1. Keyed method. The sender and the receiver possess the same key phrase. When using our algorithm, the key phrase is inputed and transformed to the required index phrase.
2. Unkeyed method. If there is no common key phrase for the sender and the receiver, the message can be used. The sending message is transformed directly to the required index phrase.

In the following sections, we shall introduce two improvements of MD5 and SHA with keyed and unkeyed methods, respectively.

5. Keyed/Unkeyed MD5

Keyed/Unkeyed MD5 can convert an arbitrary length of message to a 128-bits fixed length of Message Digest. The flowchart of Keyed MD5 and Unkeyed MD5 are described in Figs. 3 and 4, respectively. For convenience, we describe both algorithms together in one as follows.

Step 1. Input:

1.1 If a Keyed MD5 then inputs the key, its length is k bits, $k \geqq 0$.

1.2 Input the message M with length m bits, $m \geqq 0$.

Step 2. Construct index phrase: If a Keyed MD5 then converts the key to index phrase; otherwise converts the message to index phrase.

Note that the principle of conversion is: If less than 68 bytes, then appends a series of “0” at the tail of index phrase. If exceeding 68 bytes then only uses 68 bytes in the front as the index phrase. Therefore, we may get a 68-bytes array of index phrase $\boldsymbol{I}, \boldsymbol{I}[0], \boldsymbol{I}[1], \ldots, \boldsymbol{I}[67]$.

Step 3. Append k padding bits: Append padding bits $(100..0)_2$ with length k $(1 \leqq k \leqq 512$, such that $(m + k) \bmod 512 = 448)$ at M’s tail. After padding, M is constituted by the subblocks with 512 bits, each subblock is indicated as SubBlock_i $(1 \leqq i \leqq n)$. Because of SubBlock_n is not yet full, it can be divided into two parts: the front part of 448 bits is from the M’s tail and k padding bits; while the rear part is blank now.

Step 4. Append the length of M: Append the rear part of the nth subblock, SubBlock_n [449..512], with the original M’s length. This shows that M’s length is less than 2^{64}.

Step 5. Definition and initialization:

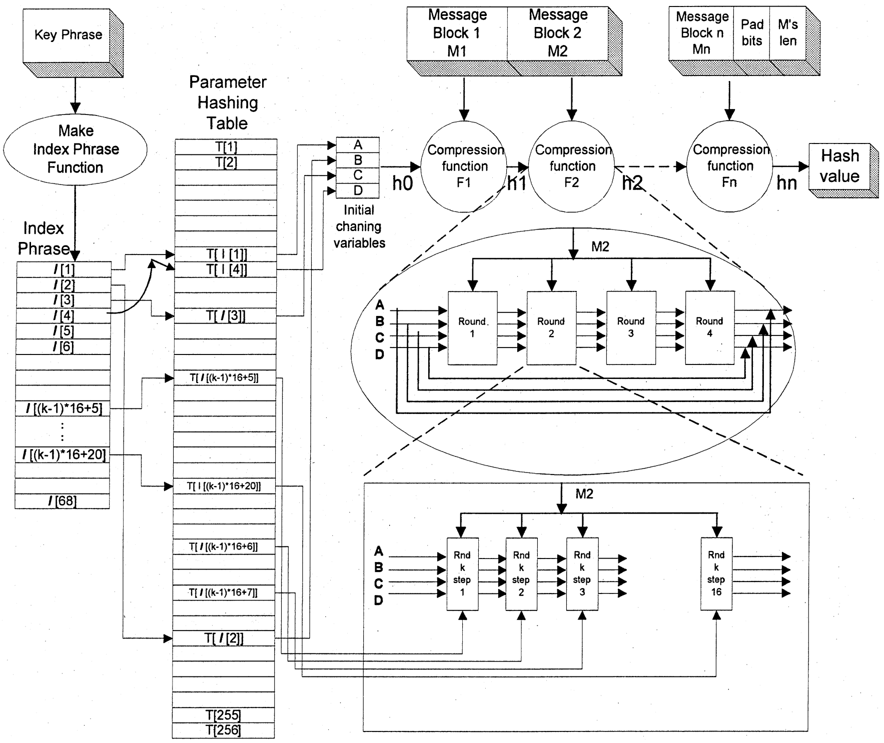

Fig. 3. Keyed MD5.

5.1 Definition:

(a) Basic functions

$$F(X,Y,Z) = (X \text{ AND } Y)\text{OR}((\neg X) \text{ AND } Z)$$
$$G(X,Y,Z) = (X \text{ AND } Z) \text{ OR } (Y \text{ AND } (\neg Z))$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \text{ OR } (\neg Z)).$$

(b) Mapping Table T: The size is 256 bytes.
Write $T[i]$: See Step 5.2.a.
Read $T[i]$: Set on a flag after an item $T[i]$ is used. When a used item is to be accessed again, a collision occurs. The simplest way to solve the problem of collision is to access the next item below that. If that item has also been used, then try its next again. Continuing the check until a unused item is found. For example, as shown in Fig. 4, $\boldsymbol{I}[4]$ and $\boldsymbol{I}[1]$ are collision, then the next index below $\boldsymbol{I}[4]$, i.e. the item $T[\boldsymbol{I}[4] + 1]$ is to be accessed.

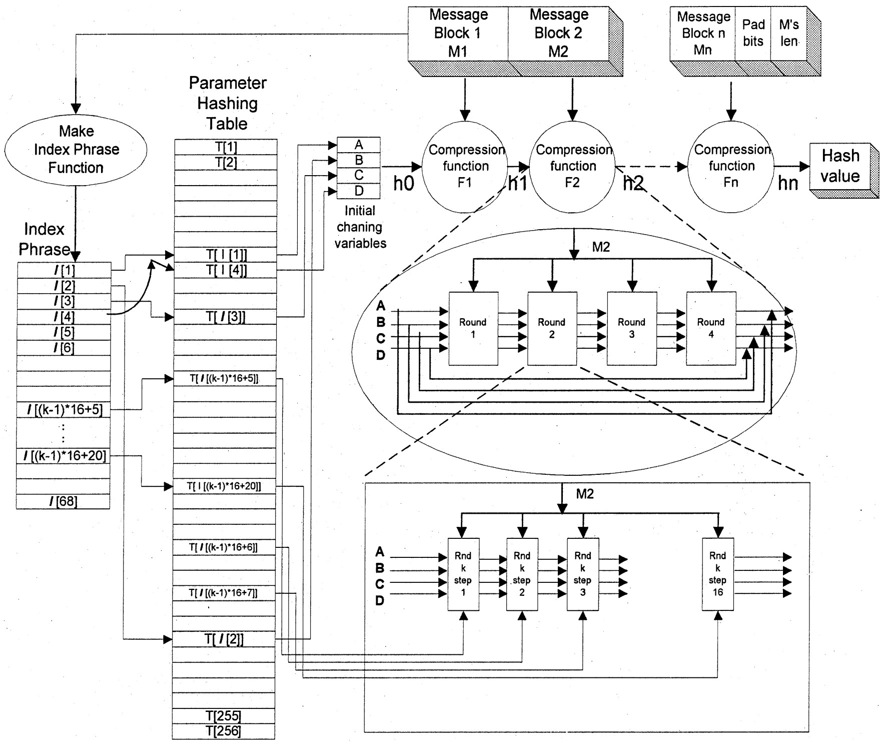

Fig. 4. Unkeyed MD5.

(c) Round functions: $T[i]$ is changed to $T[\boldsymbol{I}[i]]$ for dynamic retrieval.

$\text{Round1}(a,b,c,d,k,s,i)\{a=b+((a+F(b,c,d)+\text{SubBlock}_k+T\boldsymbol{I}[i]])\ll s)\}$

$\text{Round2}(a,b,c,d,k,s,i)\{a=b+((a+G(b,c,d)+\text{SubBlock}_k+T\boldsymbol{I}[i]])\ll s)\}$

$\text{Round3}(a,b,c,d,k,s,i)\{a=b+((a+H(b,c,d)+\text{SubBlock}_k+T\boldsymbol{I}[\text{i}]])\ll s)\}$

$\text{Round4}(a,b,c,d,k,s,i)\{a=b+((a+H(b,c,d)+\text{SubBlock}_k+T\boldsymbol{I}[i]])\ll s)\}.$

5.2 Initialization:

(a) Initialize the table: put 256 values into array $T[1..\ 256]$.

$T[i] := 2^{32} * abs(\sin(i)), 1 \leqq i \leqq 64,$

i is in radians.

(b) Initialize chaining variables: Let $\boldsymbol{I}[1..68]$ be the index phrase.

$\text{word}_{32}A = T[\boldsymbol{I}[1]]$

$\text{word}_{32}B = T[\boldsymbol{I}[2]]$

$\text{word}_{32}C = T[\boldsymbol{I}[3]]$

$\text{word}_{32}D = T[\boldsymbol{I}[4]].$

(c) Duplication: word$_{32}$ $P = A$, $Q = B$, $R = C$, $S = D$;

Step 6. Algorithm:

```
For $(inti = 0; i \leqslant n - 1; i+)$ { //Repeatedly process for each subblock, totally
$4 * 16$ steps.
  Rnd1(a, b, c, d, k, s, i)
    (ABCD 0 7 1) (DABC 1 12 2)(CDAB 2 17 3)(BCDA 3 22 4)
    (ABCD 4 7 5)(DABC 5 12 6)(CDAB 6 17 7)(BCDA 7 22 8)
    (ABCD 8 7 9)(DABC 9 12 10)(CDAB 10 17 11)(BCDA 11 22 12)
    (ABCD 12 7 13)(DABC 13 12 14)(CDAB 14 17 15)(BCDA 15 22 16)
  Rnd2(a, b, c, d, k, s, i)
    (ABCD 1 5 17) (DABC 6 9 18)(CDAB 11 14 19)(BCDA 0 20 20)
    (ABCD 5 5 21)(DABC 10 9 22)(CDAB 15 14 23)(BCDA 4 20 24)
    (ABCD 9 5 25)(DABC 14 9 26)(CDAB 3 14 27)(BCDA 8 20 28)
    (ABCD 13 5 29)(DABC 2 9 30)(CDAB 7 14 31)(BCDA 12 20 32)
  Rnd3(a, b, c, d, k, s, i)
    (ABCD 5 4 33) (DABC 8 11 34)(CDAB 11 16 35)(BCDA 14 23 36)
    (ABCD 1 4 37)(DABC 4 11 38)(CDAB 7 16 39)(BCDA 10 23 40)
    (ABCD 13 4 41)(DABC 0 11 42)(CDAB 3 16 43)(BCDA 6 23 44)
    (ABCD 9 4 45)(DABC 12 11 46)(CDAB 15 16 47)(BCDA 2 23 48)
  Rnd4(a, b, c, d, k, s, i)
    (ABCD 0 6 49)(DABC 7 10 50)(CDAB 14 15 51)(BCDA 5 21 52)
    (ABCD 12 6 53)(DABC 3 10 54)(CDAB 10 15 55)(BCDA 1 21 56)
    (ABCD 8 6 57)(DABC 15 10 58)(CDAB 6 15 59)(BCDA 13 21 60)
    (ABCD 4 6 61)(DABC 11 10 62)(CDAB 2 15 63)(BCDA 9 21 64)
} //End of loop i.
```

Step 7. Output message digest: Add the values of the last four 32-bits outputs, A, B, C, D, with their initial values $(A = A + P; B = B + Q; C = C + R; D = D + S)$ and then concatenate the four Word$_{32}$ values to obtain the M's 128-bits Message Digest.

6. Keyed/Unkeyed SHA

Keyed/Unkeyed SHA converts an arbitrary length of message to a 160-bits fixed length of Message Digest. The flowcharts of Keyed SHA and Unkeyed SHA are described in Figs. 5 and 6, respectively. For convenience, we describe both algorithms together as follows.

Step 1. Input:

1.1 If a Keyed SHA then inputs the key, its length is k bits, $k \geqq 0$.

1.2 Input the message M with length m bits, $m \geqq 0$.

Step 2. Construct index phrase:

a. If a Keyed SHA then converts the key to index phrase ***I***.

b. If an Unkeyed SHA then converts the message to index phrase.

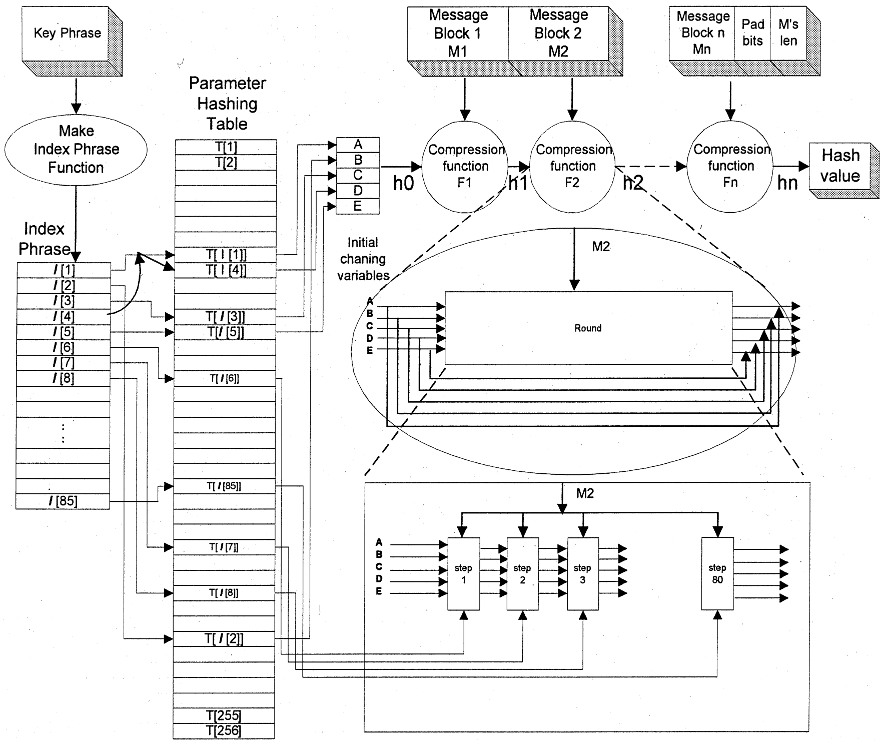

Fig. 5. Keyed SHA.

Step 3. Append k padding bits: Append the padding bits $(100..0)_2$ with length $k(1 \leqq k \leqq 512, \text{and}(m+k) \bmod 512 = 448)$ at M's tail. After padding, M is constituted by the subblocks with 512 bits, each subblock is indicated as SubBlocki $(1 \leqq i \leqq n)$. Because of SubBlock$_n$ is not yet full, it can then be divided into two parts: the front part of 448 bits is from the tail of M and the k padding bits, and the rear part of 64 bits are blanks now.

Step 4. Fill in the length of M: Fill in the rear part of 64 blanks as stated in Step 3, SubBlock$_n$[449..512], with the original M's length. This shows that M's length is less than 2^{64}.

Step 5. Definition and initialization:

5.1 Definition:

(a) Basic functions

$$F(X,Y,Z) = (X \text{ AND } Y) \text{ OR } ((\neg X) \text{ AND } Z)$$

$$G(X,Y,Z) = X \oplus Y \oplus Z$$

$$H(X,Y,Z) = (X \text{ AND } Y) \text{ OR } (X \text{ AND } Z) \text{ OR } (Y \text{ AND } Z)$$

$$I(X,Y,Z) = X \oplus Y \oplus Z.$$

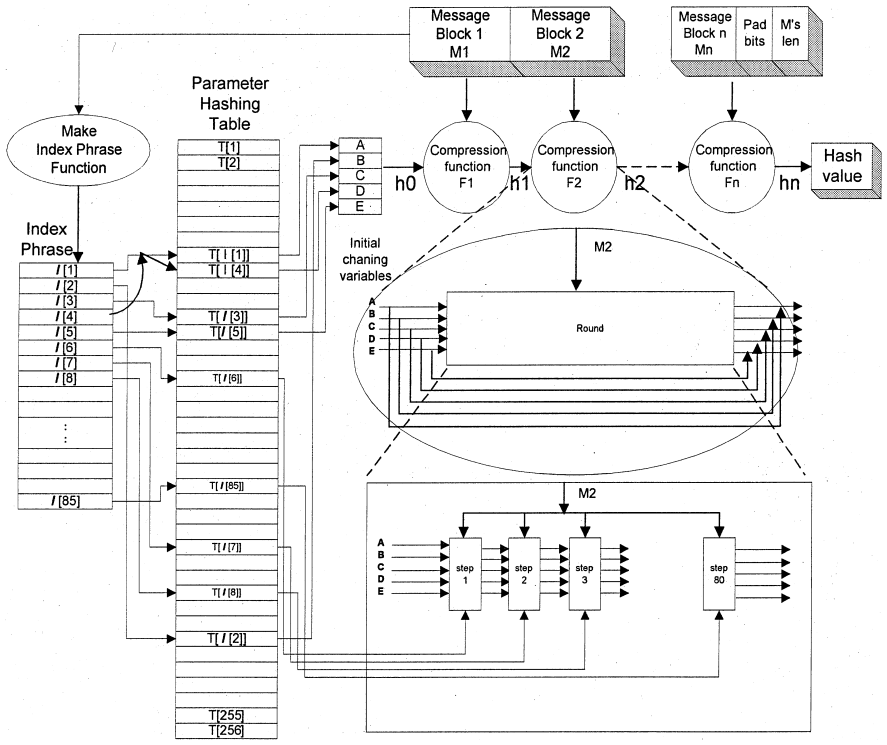

Fig. 6. Unkeyed SHA.

(b) The mapping Table *T*: The size is 256 bytes.
Write *T*[*i*]: See Step 5.2.a.
Read *T*[*i*]: As stated in Step 5.1.b of Keyed/Unkeyed MD5.
Round functions: Change *T*[*i*] to ***TI***[*i*] for dynamic retrieval.

$$\text{Round}(a,b,c,d,k,s,i)\{a = b + ((a + \mathbf{Z}(b,c,d) + X[k] + T[I[i]]) \ll s),\}$$
$$Z = F \quad 0 \leqq i \leqq 19,$$
$$Z = G \quad 20 \leqq i \leqq 39,$$
$$Z = H \quad 40 \leqq i \leqq 59,$$
$$Z = I \quad 60 \leqq i \leqq 79.$$

5.2 Initialization:
(a) Initialize the table: put 256 values into array *T*[1.. 256].

$$T[i] := 2^{32} * abs(\sin(i)), 1 \leqq i \leqq 64, \text{i is in radians.}$$

```
(b) Initialize Chaining Variables: Let $\boldsymbol{I}[1..68]$ be the index phrase.
  word$_{32}$ $A = T[\boldsymbol{I}[1]]$
  word$_{32}$ $B = T[\boldsymbol{I}[2]]$
  word$_{32}$ $C = T[\boldsymbol{I}[3]]$
  word$_{32}$ $D = T[\boldsymbol{I}[4]]$
  word$_{32}$ $E = T[\boldsymbol{I}[5]]$
(c) Duplication:
  word$_{32}$ $P = A,\ Q = B,\ R = C,\ S = D,\ V = E;$
Step 6. Algorithm:
For $(inti = 0; i \leqslant n - 1; i+)$ {//Repeatedly process for each subblock, totally 80 steps
  for $(intj = 0; j \leqslant 79; j+)$
    if $(j \leqslant 15) X[j] = \mathrm{SubBlock}_i[j]$;
    else $X[j] = (X[j-3] \oplus X[j-8] \oplus X[j-14] \oplus X[j-16])$
  for $(intj = 0; j \leqslant 79; j+)4$ {// Change$T_j$ to$T_{Rj}$
    int TEMP $= (A \ll 5) + \mathrm{Round}(a, b, c, d, k, s, i) + E + X[j] + \mathrm{T_{Rj}}$;
    $E = D; D = C; C = B \ll 30; B = A; A = \mathrm{TEMP}$;
}
} // End of loop i
```

Step 7. Output message digest: Add the values of the last five 32-bits outputs, A, B, C, D, E, with their initial values ($A = A + P$; $B = B + Q$; $C = C + R$; $D = D + S$; $E = E + V$), and then concatenate the five Word$_{32}$ to obtain M's 160-bits Message Digest.

7. Experimental results and discussion

In this section, we shall list the experimental results that show the comparisons of efficiencies among the six algorithms: MD5, Keyed-MD5, Unkeyed-MD5, SHA, Keyed-SHA, and Unkeyed-SHA. The methodology of experiment is as follows. Download the MD5 and SHA program from the Internet, modify them to My-MD5 and My-SHA programs. The former program can execute three types of algorithms, MD5, Keyed-MD5 or Unkeyed-MD5 by choosing a proper parameter. Similarly, the latter program can also execute three types of algorithms, SHA, Keyed-SHA or Unkeyed-SHA by specifying the proper parameter.

The machine we used is a PC Intel Pentium 166, the operating system is Windows 95, and the programming language is Turbo C+ 3.0. We list the execution result as follows:

1. To run MD5, the average efficiency is 144,871 (byte/s).
2. To run Unkeyed MD5, the average efficiency is 143,047 (byte/s).
3. To run Keyed MD5, the average efficiency is 122,399 (byte/s).
4. To run SHA, the average efficiency is 87,903 (byte/s).

5. To run Unkeyed SHA, the average efficiency is 87,593 (byte/s).
6. To run Keyed SHA, the average efficiency is 80,958 (byte/s).

As stated above, we can see that the improved algorithms are slower than the original ones with 1% or so. The reason is that in our programs the function of "BuildKeyPhrase" will spend some CPU time to build the index phrase. From this, the proposed algorithms will provide better safety and not degrade the efficiency too much.

Finally, there are two phenomena in the proposed algorithms:

1. When the key phrase is shorter than the index phrase, our method is to append 0's. This will produce a series of collisions and a situation of sequential retrievals of parameters.
2. When the key phrase is longer than the index phrase, we cut out only k bytes in the front.

These two problems can be solved through designing better mapping functions for the algorithms.

References

[1] E. Biham, A. Shamir, Differential analysis of deslike cryptosystems, Advances in Cryptology 2-21.
[2] E. Biham, A. Shamir, Differential cryptanalysis of feal and N-hash, Advances in Cryptology — EUROCRYP'91 Proceedings, 1991, pp. 1–16.
[3] ThomasA. Berson, Differential Cryptanalysis Mod 232 with Applications to MD5, Advances in Cryptology-EUROCYPT'92 Proceedings, 1992, pp. 71–80.
[4] R. L. Rivest, The MD5 Message Digest Algorithm, RFC 1321, 1992.
[5] E. Biham, A. Shamir, Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI, and Lucifer, Advances in Cryptology – CRYPTO'91, pp. 156–171.
[6] National Institute of Standards and Technology, NIST FIPS PUB 180, Secure Hash Standard, US Department of Commerce, 1993.
[7] Y. Zheng, J. Pieprzyk, J. Seberry, HAVAL-a one-way hashing algorithm with variable length of output, Advances in Cryptology-AUSCRYPT '92. Workshop on the Theory and Application of Cryptographic Techniques Proceedings, 1993, pp. 83–104.
[8] Bertden. Boer, A. Bosselaers, Collisions for the Compression Function of MD5, Advances in Cryptology-EROCRYPT'93 Proceedings, Springer, Berlin, 1994, pp. 293–304.
[9] W. Stallings, Sha the secure hash algorithm, Dr. Dobb's Journal 4 (1994) 32–34.
[10] B. Kaliski, M. Robshaw, Message Authentication with MD5, RSA Labs' CryptoBytes, vol. 1, No. 1, Springer, Berlin, 1995 (http://www.rsa.com/relabs/cryptobytes/).
[11] National Institute of Standards and Technology, NIST FIPS PUB 180-1, Secure Hash Standard, US Department of Commerce, 1995.
[12] R. Anderson, E. Biham, Tiger: a fast new hash function, Fast Software Encryption. Third International Workshop Proceedings, 1996, pp. 89–97.
[13] M. Bellare, R. Canetti, H. Krawczyk, The HMAC Construction, RSA Laboratories's CryptoBytes, vol. 2, No. 1, 1996.
[14] H. Dobbertin, Cryptanalysis of MD4, Fast Software Encryption. Third International Workshop Proceedings, 1996, pp. 53–69.
[15] H. Dobbertin, The status of MD5 after a recent attack, RSA Labs' CryptoBytes, vol. 2, No. 2, Summer 1996. http://www.rsa.com/rsalabs/pubs/cryptobytes.html.

[16] H. Dobbertin, Cryptanalysis of MD5 Compress, 2 May 1996. http://www.ph.tn.tudelft.nl/visser/hashes.html.

[17] Hans Dobbertin, Antoon Bosselaers, Bart Preneel, RIPEMD-160: a strengthened version of RIPEMD, Fast Software Encryption, Third International Workshop Proceedings, 1996, pp. 71–82.

[18] H. Krawczyk, M. Bellare, R. Canetti, HMAC-MD5: Keyed-MD5 for Message Authentication, submitted for RFC (Request For Comments) publication, August, 1996.

[19] B. Schneier, Applied Cryptography-Protocols Algorithms and Source Code in C, Wiley, New York, 1996.

[20] A. Bosselaers, H. Dobbertin, B. Preneel, The RIPEMD-160 cryptographic hash function (includes related article with sample problem) (Technology Information), Dr. Dobb's Journal, vol. 22, No. 1, January 1997, p. 24.