密码散列函数
安全散列函数
hash：
音译：哈希
意译：散列，杂凑

Cryptographic Hash Functions

In this chapter, we elaborate on cryptographic hash functions. More specifically, we introduce the basic principles and properties of such functions in Section 8.1, address a basic construction (i.e., the Merkle-Damgård construction) in Section 8.2, overview exemplary cryptographic hash functions in Section 8.3, and conclude with some final remarks in Section 8.4.

8.1 INTRODUCTION

一般而言，原像空间大小远大于像空间大小

As mentioned in Section 2.1.2 and formally expressed in Defintion 2.3, a *hash function* is an efficiently computable function $h : \Sigma_{in}^* \rightarrow \Sigma_{out}^n$ that takes an arbitrarily sized[1] input word $x \in \Sigma_{in}^*$ (with Σ_{in} representing the input alphabet) and generates an output word $y \in \Sigma_{out}^n$ (with Σ_{out} representing the output alphabet) of size n. Furthermore, a *cryptographic hash function* is a hash function that has specific properties. There are basically three properties that are relevant from a cryptographic viewpoint.

抗原像攻击

- A hash function h is *preimage resistant* if it is computationally infeasible to find an input word $x \in \Sigma_{in}^*$ with $h(x) = y$ for a given (and randomly chosen) output word $y \in_R \Sigma_{out}^n$.

抗第二原像攻击、抗弱碰撞攻击

- A hash function h is *second-preimage resistant* or *weak collision resistant* if it is computationally infeasible to find a second input word $x' \in \Sigma_{in}^*$ with $x' \neq x$ and $h(x') = h(x)$ for a given (and randomly chosen) input word $x \in_R \Sigma_{in}^*$.

1 Remember from Section 2.1.2 that one usually has to assume a maximum length n_{max} for input words. In this case, the hash function is formally expressed as $h : \Sigma_{in}^{n_{max}} \rightarrow \Sigma_{out}^n$.

抗碰撞攻击，抗强碰撞攻击

- A hash function h is *collision resistant* or *strong collision resistant* if it is computationally infeasible to find two input words $x, x' \in \Sigma_{in}^*$ with $x' \neq x$ and $h(x') = h(x)$.

There are some comments to make at this point:

- In some literature, collision resistant hash functions are also called *collision free*. This term is inappropriate, because collisions always occur if one uses hash functions (i.e., functions that hash arbitrarily sized arguments to a fixed size). Consequently, the term collision free is not used as an attribute to cryptographic hash functions in this book.
- In a complexity-theoretic setting, one cannot say that finding a collision for a given hash function is a difficult problem. In fact, finding a collision (for a given hash function) is a problem instance rather than a problem (refer to Section 6.2 for a discussion about the difference between a problem and a problem instance). This is because there is always an efficient algorithm that finds a collision, namely one that simply outputs two input words that hash to the same value. Thus, the concept of collision resistance only makes sense if one considers a sufficiently large class (or family) of hash functions from which one is chosen at random. An algorithm to find collisions must then work for all hash functions of the class, including the one that is chosen at random.

CRF must be WCRF, but not vice versa

- A collision resistant hash function must be second-preimage resistant. Otherwise it is possible to find a second preimage for an arbitrarily chosen input word, and this second preimage then yields a collision. The converse, however, is not true—that is, a second-preimage resistant hash function must not be collision resistant (that's why we used the terms *weak collision resistant* and *strong collision resistant* in the first place). Consequently, collision resistance implies second-preimage resistance, but not vice versa.
- A (strong or weak) collision resistant hash function must not be preimage resistant. For example, let g be a collision resistant hash function with an n-bit output and h a pathological $(n+1)$-bit hash function that is defined as follows:[2]

$$h(x) = \begin{cases} 1 \,||\, x & \text{if } |\, x \,| = n \\ 0 \,||\, g(x) & \text{otherwise} \end{cases}$$

2 This example is taken from Ueli Maurer's seminar entitled "Cryptography—Fundamentals and Applications."

On the one hand, h is collision resistant. If $h(x)$ begins with a one, then there is no collision at all. If $h(x)$ begins with a zero, then finding a collision means finding a collision for g (which is assumed to be computationally infeasible). On the other hand, h is not preimage resistant. For all $h(x)$ that begin with a one, it is trivial to find a preimage (just drop the leading one) and to invert h accordingly. Consequently, h is a hash function that is collision resistant but not preimage resistant, and we conclude that preimage resistance and collision resistance are inherently different properties that must be distinguished accordingly.

In practice, Σ_{in} and Σ_{out} are often set to $\{0,1\}$, and a hash function then represents a map from $\{0,1\}^*$ to $\{0,1\}^n$. A question that occurs immediately is how large to choose the parameter n. A lower bound for n is obtained by the *birthday attack*. This attack is based on the *birthday paradox* that is well known in probability theory. It says that the probability of two persons in a group sharing the same birthday is greater than $1/2$ if the group is chosen at random and has more than 23 members. To obtain this result, we employ a sample space Σ that consists off all n-tuples over the 365 days of the year (i.e., $\mid \Sigma \mid = 365^n$). Let $\Pr[\mathcal{A}]$ be the probability that at least two out of n persons have the same birthday. This value is difficult to compute directly. It is much simpler to compute $\Pr[\overline{\mathcal{A}}]$ (i.e., the probability that all persons have different birthdays) and to derive $\Pr[\mathcal{A}]$ from this value. In fact, $\Pr[\mathcal{A}]$ can be computed as follows (for $0 \leq n \leq 365$):

$$\begin{aligned}
\Pr[\mathcal{A}] &= 1 - \Pr[\overline{\mathcal{A}}] \\
&= 1 - \frac{\mid \overline{\mathcal{A}} \mid}{\mid \Sigma \mid} \\
&= 1 - 365 \cdot 364 \cdot \ldots \cdot (365 - n) \cdot \frac{1}{365^n} \\
&= 1 - \frac{365!}{(365-n)!} \cdot \frac{1}{365^n} \\
&= 1 - \frac{365!}{(365-n)!365^n}
\end{aligned}$$

$\Pr[\mathcal{A}]$ is equal to 1 for $n > 365$ (in this case, it is not possible that all n persons have different birthdays). In either case, $\Pr[\mathcal{A}]$ grows surprisingly fast and n must only be 23 to reach a probability greater or equal to $1/2$. If $n = 23$, then

$$\Pr[\mathcal{A}] = 1 - \frac{365!}{(365-23)!365^{23}}$$

$$\begin{aligned} &= 1 - \frac{365!}{(342)!365^{23}} \\ &= 1 - \frac{365 \cdot 364 \cdots 343}{365^{23}} \approx 0.508 \end{aligned}$$

This result is somehow paradoxical. If we fix a date and ask for the number of persons that are required to make the probability that at least one person has this date as his or her birthday, then n must be much larger. In fact, in this case n has to be $\lceil 362/2 \rceil = 183$.

Applying this argument to hash functions means that finding two persons with the same birthday reveals a collision, whereas finding a person with a given birthday reveals a second preimage. Hence, due to the birthday paradox, one can argue that collision resistance is much more difficult to achieve than second-preimage resistance. More specifically, one can show that for any collision resistant hash function with an n-bit output, no attack finding a collision betters a birthday attack with a worst-case running time of

$$O(\sqrt{2^n}) = O(2^{n/2})$$

That's why the birthday attack is sometimes also referred to as *square root attack*. This result implies that a collision resistant hash function must produce outputs that are twice as long as one would usually suggest to make an exhaustive search computationally infeasible. For example, if we assume that searching a key space of 2^{64} is computationally infeasible, then we must use a hash function that outputs at least $2 \cdot 64 = 128$ bits.

In addition to preimage, second-preimage, and collision resistance, there are also some other properties of hash functions mentioned (and sometimes discussed) in the literature.

- A hash function h is *noncorrelated* if its input bits and output bits are not correlated in one way or another.
- A hash function h is *generalized collision resistant* if it is computationally infeasible to find two input words x and x' with $x' \neq x$ such that $h(x)$ and $h(x')$ are similar in some specific sense (e.g., they are equal in some bits).
- A hash function h is *weakened collision resistant* if it is computationally infeasible to find two input words x and x' with $x' \neq x$ and $h(x) = h(x')$ such that x and x' are similar in some specified sense (e.g., they are equal in some bits).

These properties are just mentioned for completeness; they are not further used in this book.

Having the previously mentioned properties (i.e., preimage, second-preimage, and collision resistance) in mind, one can define *one-way hash functions* (OWHFs), *collision resistant hash functions* (CRHFs), and *cryptographic hash functions* as suggested in Definitions 8.1 and 8.2.

Definition 8.1 (One-way hash function) *An* OWHF *is a hash function* $h : \Sigma_{in}^{*} \rightarrow \Sigma_{out}^{n}$ *that is preimage resistant and second-preimage resistant.*

Definition 8.2 (Collision resistant hash function) *A* CRHF *is a hash function* $h : \Sigma_{in}^{*} \rightarrow \Sigma_{out}^{n}$ *that is preimage resistant and collision resistant.*

Note that a CRHF is always an OWHF (whereas the converse may not be true). Also note that alternative terms sometimes used in the literature are *weak one-way hash functions* for OWHFs and *strong one-way hash functions* for CRHFs. As suggested in Definition 2.4, we use the term *cryptographic hash function* to refer to either of them.

8.2 MERKLE-DAMGÅRD CONSTRUCTION

Most cryptographic hash functions in use today follow a construction that was independently proposed by Ralph C. Merkle and Ivan B. Damgård in the late 1980s [1, 2].[3] According to their construction, an *iterated hash function* h is computed by repeated application of a collision resistant compression function $f : \Sigma^{m} \longrightarrow \Sigma^{n}$ with $m, n \in \mathbb{N}$ and $m > n$ to successive blocks $x_1, \ldots, x_n$ of a message x.[4]

As illustrated in Figure 8.1, the compression function f takes two input arguments:

1. A b-bit message block;
2. An l-bit chaining value (sometimes referred to as H_i for $i = 0, \ldots, n$).

In a typical setting, l is 128 or 160 bits and b is 512 bits. The output of the compression function can be used as a new l-bit chaining value, which is input to the next iteration of the compression function. Referring to the notation introduced earlier, $m = b + l$ and $n = l$.

3 Both papers were presented at CRYPTO '89.

4 Note that the input alphabet Σ_{in} and the output alphabet Σ_{out} are assumed to be the same (denoted as Σ).

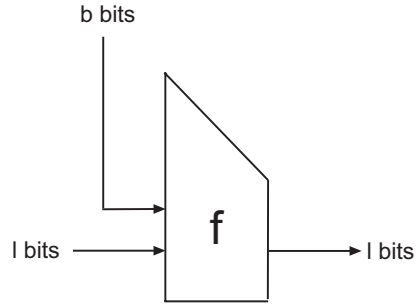

Figure 8.1 A compression function f.

Against this background, an iterated hash function h can then be constructed, as illustrated in Figure 8.2. In this figure, f represents the compression function and g represents an output function.[5] The message x is padded to a multiple of b bits and divided into a sequence of n b-bit message blocks $x_1, \ldots, x_n$. The compression function f is then repeatedly applied, starting with an initial value ($IV = H_0$) and the first message block x_1, and continuing with each new chaining value H_i and successive message block x_{i+1} for $i = 1, \ldots, n-1$. After the last message block x_n has been processed, the final chaining value H_n is subject to the output function g, and the output of this function is the output of the iterated hash function h for x (i.e., $h(x)$).

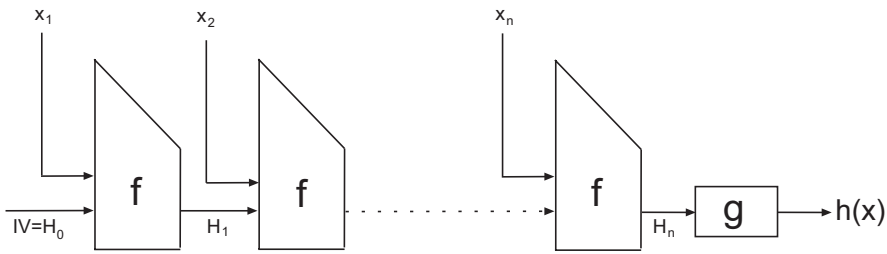

Figure 8.2 An iterated hash function h.

Hence, an iterative hash function h for $x = x_1 x_2 \ldots x_n$ can be recursively computed according to the following set of equations:

$$H_0 = IV$$

5 It goes without saying that g can also be the identity function.

$$\begin{aligned} H_i &= f(H_{i-1}, x_i) \text{ for } i = 1, \ldots, n \\ h(x) &= g(H_n) \end{aligned}$$

As mentioned earlier, the message to be hashed must be padded to a multiple of b bits. One possibility is to pad x with zeros. Padding with zeros, however, may also introduce ambiguity about x. For example, the message 101110 padded to 8 bits would be 10111000 and it is then unclear how many trailing zeros were present in the original message. Several methods are available to resolve this problem. Merkle proposed to append the bit length of x at the end of x. To make the additional length field easy to find, it is right-justified in the final block. Following this proposal, the padding method of choice in currently deployed hash functions is to append a one, a variable number of zeros, and the binary encoding of the length of the original message to the end of the message.

Merkle and Damgård showed that in their construction, finding a collision for h (i.e., finding two input words x and x' with $x \neq x'$ and $h(x) = h(x')$) is at least as hard as finding a collision for the underlying compression function f. This also means that if f is a collision resistant compression function, and h is an iterated hash function making use of f, then h is a cryptographic hash function that is also collision resistant. Put in other words, the iterated hash function inherits the collision resistance property from the underlying compression function.

In the literature, there are many proposals for collision resistant compression functions that can be turned into collision resistant cryptographic hash functions according to the Merkle-Damgård construction. Some examples can, for example, be found in [1, 2].

8.3 EXEMPLARY CRYPTOGRAPHIC HASH FUNCTIONS

The driving force for cryptographic hash functions was public key cryptography in general, and digital signature systems in particular. Consequently, the company RSA Security, Inc., played a crucial role in the development and deployment of many practically relevant cryptographic hash functions. The first cryptographic hash function developed by RSA Security, Inc., was acronymed MD (standing for *message digest*). It was proprietary and never published. MD2 specified in RFC 1319 [3] was the first published cryptographic hash function in widespread use (it was, for example, used in the secure messaging products of RSA Security, Inc.). When Merkle proposed a cryptographic hash function called SNEFRU that was several

times faster than MD2,[6] RSA Security, Inc., responded to the challenge with MD4[7] specified in RFC 1320 [4] (see Section 8.3.1). MD4 took advantage of the fact that newer processors could do 32-bit operations, and it was therefore able to be faster than SNEFRU. In 1991, SNEFRU and some other cryptographic hash functions were successfully attacked[8] using differential cryptanalysis [5]. Furthermore, some weaknesses were found in a version of MD4 with two rounds instead of three [6]. This did not officially break MD4, but it made RSA Security, Inc., sufficiently nervous that it was decided to strengthen MD4. MD5 was designed and specified in RFC 1320 [7] (see Section 8.3.2). MD5 is assumed to be more secure than MD4, but it is also a little bit slower. Due to some recent results, MD4 must be considered to be insecure [8], and MD5 must be considered to be partially broken [9].[9] In 2004, a group of Chinese researchers found and published collisions for MD4, MD5, and some other cryptographic hash functions.[10] Nevertheless, MD4 and MD5 are still useful study objects for the design principles of cryptographic hash functions.

Table 8.1
Secure Hash Algorithms as Specified in FIPS 180-2

Algorithm	Message Size	Block Size	Word Size	Hash Value Size
SHA-1	$< 2^{64}$ bits	512 bits	32 bits	160 bits
SHA-224	$< 2^{64}$ bits	512 bits	32 bits	224 bits
SHA-256	$< 2^{64}$ bits	512 bits	32 bits	256 bits
SHA-384	$< 2^{128}$ bits	1,024 bits	64 bits	384 bits
SHA-512	$< 2^{128}$ bits	1,024 bits	64 bits	512 bits

In 1993, the U.S. NIST proposed the *Secure Hash Algorithm* (SHA), which is similar to MD5, but even more strengthened and also a little bit slower. Probably after discovering a never-published weakness in the orginal SHA proposal,[11] the NIST revised it and called the revised version SHA-1. As such, SHA-1 is specified in the Federal Information Processing Standards Publication (FIPS PUB) 180-1 [12],[12] also known as *Secure Hash Standard* (SHS). In 2002, FIPS PUB 180 was revised

6 The function was proposed in 1990 in a Xerox PARC technical report entitled *A Software One Way Function*.

7 There was an MD3 cryptographic hash function, but it was superseded by MD4 before it was ever published or used.

8 The attack was considered successful because it was shown how to systematically find a collision (i.e., two messages with the same hash value).

9 One problem with MD5 is that the compression function is known to have collisions (e.g., [10]).

10 http://eprint.iacr.org/2004/199.pdf

11 At CRYPTO '98, Florent Chabaud and Antoine Joux published a weakness of SHA-0 [11]. This weakness was fixed by SHA-1, so it is reasonable to assume that they found the original weakness.

12 SHA-1 is also specified in informational RFC 3174 [13].

Table 8.2
Truth Table of the Logical Functions Employed by MD4

X	Y	Z	f	g	h
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	1	1	1

a second time and the resulting FIPS PUB 180-2[13] superseded FIPS PUB 180-1 beginning February 1, 2003. In addition to superseding FIPS 180-1, FIPS 180-2 also added three new algorithms that produce and output larger hash values (see Table 8.1). The SHA-1 algorithm specified in FIPS 180-2 is the same algorithm as specified in FIPS 180-1, although some of the notation has been modified to be consistent with the notation used in SHA-256, SHA-384, and SHA-512. As summarized in Table 8.1, SHA-1, SHA-256, SHA-384, and SHA-512 produce and output hash values of different sizes (160, 256, 384, and 512 bits), and their maximal message sizes, block sizes, and word sizes also vary considerably. In February 2004, the NIST published a change notice for FIPS 180-2 to include SHA-224.[14] SHA-224 is identical to SHA-256, but uses different initial hash values and truncates the final hash value to its leftmost 224 bits. It is also included in Table 8.1.

In addition to the cryptographic hash functions proposed by RSA Security, Inc., and the NIST, there are at least two competing proposals developed entirely in Europe (i.e., RIPEMD-128 and RIPEMD-160 [15, 16]). These cryptographic hash functions are not further addressed in this book.

MD4, MD5, and RIPEMD-128 produce hash values of 128 bits, whereas RIPEMD-160 and SHA-1 produce hash values of 160 bits. The newer versions of SHA produce hash values that are even longer. From a security viewpoint, long hash values are preferred (because they reduce the likelihood of collisions in the first place). Consequently, it is recommended to replace MD5 with SHA-1 (or any other hash function from the SHA family) where possible and appropriate. MD4, MD5, and SHA-1 are overviewed and discussed next.

13 http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf
14 SHA-224 is also specified in informational RFC 3874 [14].

Algorithm 8.1 The MD4 hash function (overview).

$(m = m_0 m_1 \dots m_{s-1})$

Construct $M = M[0]M[1] \dots M[N-1]$
$A \leftarrow$ `0x67452301`
$B \leftarrow$ `0xEFCDAB89`
$C \leftarrow$ `0x98BADCFE`
$D \leftarrow$ `0x10325476`
for $i = 0$ to $N/16 - 1$ do
    for $j = 0$ to 15 do $X[j] = M[i \cdot 16 + j]$
    $A' \leftarrow A$
    $B' \leftarrow B$
    $C' \leftarrow C$
    $D' \leftarrow D$
    Round 1 (Algorithm 8.2)
    Round 2 (Algorithm 8.3)
    Round 3 (Algorithm 8.4)
    $A \leftarrow A + A'$
    $B \leftarrow B + B'$
    $C \leftarrow C + C'$
    $D \leftarrow D + D'$

$(h(m) = A \parallel B \parallel C \parallel D)$

8.3.1 MD4

As mentioned earlier, MD4 was proposed in 1990 and is specified in RFC 1320 [4].[15] It represents a Merkle-Damgård construction that hashes a message in 512-bit blocks (i.e., $b = 512$) and that produces an output of 128 bits (i.e., $l = 128$). As also mentioned earlier, MD4 was designed to be efficiently implementable on 32-bit processors. It assumes a little-endian architecture, meaning that a 4-byte word $a_1a_2a_3a_4$ represents the following integer:

$$a_4 2^{24} + a_3 2^{16} + a_2 2^8 + a_1$$

In a big-endian architecture, the same 4-byte word $a_1a_2a_3a_4$ would represent the integer

$$a_1 2^{24} + a_2 2^{16} + a_3 2^8 + a_4.$$

15 The original version of MD4 was published in October 1990 in RFC 1196. A slightly revised version of it was published in April 1992 (at the same time as MD5) in RFC 1320.

Algorithm 8.2 Round 1 of the MD4 hash function.

1. $A \leftarrow (A + f(B, C, D) + X[0]) \hookleftarrow 3$
2. $D \leftarrow (D + f(A, B, C) + X[1]) \hookleftarrow 7$
3. $C \leftarrow (C + f(D, A, B) + X[2]) \hookleftarrow 11$
4. $B \leftarrow (B + f(C, D, A) + X[3]) \hookleftarrow 19$
5. $A \leftarrow (A + f(B, C, D) + X[4]) \hookleftarrow 3$
6. $D \leftarrow (D + f(A, B, C) + X[5]) \hookleftarrow 7$
7. $C \leftarrow (C + f(D, A, B) + X[6]) \hookleftarrow 11$
8. $B \leftarrow (B + f(C, D, A) + X[7]) \hookleftarrow 19$
9. $A \leftarrow (A + f(B, C, D) + X[8]) \hookleftarrow 3$
10. $D \leftarrow (D + f(A, B, C) + X[9]) \hookleftarrow 7$
11. $C \leftarrow (C + f(D, A, B) + X[10]) \hookleftarrow 11$
12. $B \leftarrow (B + f(C, D, A) + X[11]) \hookleftarrow 19$
13. $A \leftarrow (A + f(B, C, D) + X[12]) \hookleftarrow 3$
14. $D \leftarrow (D + f(A, B, C) + X[13]) \hookleftarrow 7$
15. $C \leftarrow (C + f(D, A, B) + X[14]) \hookleftarrow 11$
16. $B \leftarrow (B + f(C, D, A) + X[15]) \hookleftarrow 19$

Let $m = m_0 m_1 \dots m_{s-1}$ be an s-bit message that is to be hashed with MD4. In a first step, an array

$$M = M[0]M[1] \dots M[N-1]$$

is constructed, where each $M[i]$ represents a 32-bit word and $N \equiv 0 \bmod 16$. Consequently, the length of M equals a multiple of $32 \cdot 16 = 512$ bits. It is constructed in two steps:

- First, the message m is padded so that its bit length is congruent to 448 modulo 512. Therefore, a single one is appended, and then zero bits are appended so that the bit length of the padded message becomes congruent to 448 modulo 512 (i.e., at least one bit and at most 512 bits must be appended). Note that padding is always performed, even if the length of the message is already congruent to 448 modulo 512. Also note that the padded message is 64 bits short of being a multiple of 512 bits.
- Second, a 64-bit binary representation of s (i.e., the length of the original message before the padding bits were added) is appended to the result of the first step. In the unlikely case that s is greater than 2^{64}, then only the low-order 64 bits of s are used (i.e., s is computed modulo 2^{64}). In either case, the 64 bits fill up the last message block from 448 to 512 bits.

Algorithm 8.3 Round 2 of the MD4 hash function.

1. $A \leftarrow (A + g(B, C, D) + X[0] + c_1) \hookleftarrow 3$
2. $D \leftarrow (D + g(A, B, C) + X[4] + c_1) \hookleftarrow 5$
3. $C \leftarrow (C + g(D, A, B) + X[8] + c_1) \hookleftarrow 9$
4. $B \leftarrow (B + g(C, D, A) + X[12] + c_1) \hookleftarrow 13$
5. $A \leftarrow (A + g(B, C, D) + X[1] + c_1) \hookleftarrow 3$
6. $D \leftarrow (D + g(A, B, C) + X[5] + c_1) \hookleftarrow 5$
7. $C \leftarrow (C + g(D, A, B) + X[9] + c_1) \hookleftarrow 9$
8. $B \leftarrow (B + g(C, D, A) + X[13] + c_1) \hookleftarrow 13$
9. $A \leftarrow (A + g(B, C, D) + X[2] + c_1) \hookleftarrow 3$
10. $D \leftarrow (D + g(A, B, C) + X[6] + c_1) \hookleftarrow 5$
11. $C \leftarrow (C + g(D, A, B) + X[10] + c_1) \hookleftarrow 9$
12. $B \leftarrow (B + g(C, D, A) + X[14] + c_1) \hookleftarrow 13$
13. $A \leftarrow (A + g(B, C, D) + X[3] + c_1) \hookleftarrow 3$
14. $D \leftarrow (D + g(A, B, C) + X[7] + c_1) \hookleftarrow 5$
15. $C \leftarrow (C + g(D, A, B) + X[11] + c_1) \hookleftarrow 9$
16. $B \leftarrow (B + g(C, D, A) + X[15] + c_1) \hookleftarrow 13$

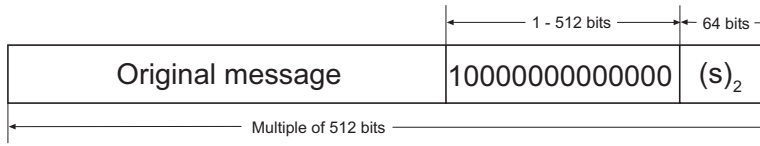

Figure 8.3 The structure of a message preprocessed to be hashed using MD4.

At this point, the resulting message has a structure as illustrated in Figure 8.3. It has a length that is an exact multiple of 512 bits. Consequently, it can be broken up into 32-bit words, and the resulting number of words (i.e., N) is still divisible by 16.

A 128-bit MD4 hash value can be constructed using Algorithm 8.1. In short, the hash value is constructed as the concatenation of four words (or registers) A, B, C, and D. First, the array M is constructed as discussed earlier, and the four registers are initialized with constant values. The array M is then processed iteratively. In each iteration, 16 words of M are taken and stored in an array X. The values of the four registers are stored for later reuse. In the main part of the algorithm, three rounds of hashing are performed (i.e., Round 1, Round 2, and Round 3). Each round consists of one operation on each of the 16 words in X (described later). The operations done in the three rounds produce new values for the four registers. Finally, the four registers are updated by adding back the values that were stored previously (the addition is always performed modulo 2^{32}).

Algorithm 8.4 Round 3 of the MD4 hash function.

1. $A \leftarrow (A + h(B, C, D) + X[0] + c_2) \hookleftarrow 3$
2. $D \leftarrow (D + h(A, B, C) + X[8] + c_2) \hookleftarrow 9$
3. $C \leftarrow (C + h(D, A, B) + X[4] + c_2) \hookleftarrow 11$
4. $B \leftarrow (B + h(C, D, A) + X[12] + c_2) \hookleftarrow 15$
5. $A \leftarrow (A + h(B, C, D) + X[2] + c_2) \hookleftarrow 3$
6. $D \leftarrow (D + h(A, B, C) + X[10] + c_2) \hookleftarrow 9$
7. $C \leftarrow (C + h(D, A, B) + X[6] + c_2) \hookleftarrow 11$
8. $B \leftarrow (B + h(C, D, A) + X[14] + c_2) \hookleftarrow 15$
9. $A \leftarrow (A + h(B, C, D) + X[1] + c_2) \hookleftarrow 3$
10. $D \leftarrow (D + h(A, B, C) + X[9] + c_2) \hookleftarrow 9$
11. $C \leftarrow (C + h(D, A, B) + X[5] + c_2) \hookleftarrow 11$
12. $B \leftarrow (B + h(C, D, A) + X[13] + c_2) \hookleftarrow 15$
13. $A \leftarrow (A + h(B, C, D) + X[3] + c_2) \hookleftarrow 3$
14. $D \leftarrow (D + h(A, B, C) + X[11] + c_2) \hookleftarrow 9$
15. $C \leftarrow (C + h(D, A, B) + X[7] + c_2) \hookleftarrow 11$
16. $B \leftarrow (B + h(C, D, A) + X[15] + c_2) \hookleftarrow 15$

The three rounds used in the MD4 hash function are different. The following operations are employed in the three rounds (X and Y denote input words, and each operation produces an output word):

$X \wedge Y$	Bitwise and of X and Y (AND)
$X \vee Y$	Bitwise or of X and Y (OR)
$X \oplus Y$	Bitwise exclusive or of X and Y (XOR)
$\neg X$	Bitwise complement of X (NOT)
$X + Y$	Integer addition of X and Y modulo 2^{32}
$X \hookleftarrow s$	Circular left shift of X by s positions $(0 \leq s \leq 31)$

Note that all of these operations are very fast and that the only arithmetic operation is addition modulo 2^{32}. As mentioned earlier, MD4 assumes a little-endian architecture.[16] Consequently, if an MD4 hash value must be computed on a big-endian machine, then the addition operation is a little bit more involved and must be implemented accordingly.

Rounds 1, 2, and 3 of the MD4 hash algorithm use the following three auxiliary functions f, g, and h:

$$f(X, Y, Z) \quad = \quad (X \wedge Y) \vee ((\neg X) \wedge Z)$$

16 Rivest chose to assume a little-endian architecture mainly because he observed that big-endian architectures are generally faster and can therefore better afford the processing penalty (of reversing each word for processing).

$$\begin{aligned} g(X,Y,Z) &= (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z) \\ h(X,Y,Z) &= X \oplus Y \oplus Z \end{aligned}$$

Each function takes as input three 32-bit words and produces as output a 32-bit word. The truth table is illustrated in Table 8.2 on page 203. The function f is sometimes known as the *selection function*, because if the n^{th} bit of X is 1, then it selects the n^{th} bit of Y for the n^{th} bit of the output. Otherwise (i.e., if the n^{th} bit of X is 0), it selects the n^{th} bit of Z for the n^{th} bit of the output. The function g is sometimes known as the *majority function*, because the n^{th} bit of the output is 1 if and only if at least two of the three input words' n^{th} bits are 1. Last but not least, the function h simply adds all input words modulo 2.

Algorithm 8.5 The MD5 hash function (overview).

$(m = m_0 m_1 \ldots m_{s-1})$
Construct $M = M[0]M[1] \ldots M[N-1]$
$A \leftarrow$ `0x67452301`
$B \leftarrow$ `0xEFCDAB89`
$C \leftarrow$ `0x98BADCFE`
$D \leftarrow$ `0x10325476`
for $i = o$ to $N/16 - 1$ do
    for $j = 0$ to 15 do $X[j] = M[i \cdot 16 + j]$
    $A' \leftarrow A$
    $B' \leftarrow B$
    $C' \leftarrow C$
    $D' \leftarrow D$
    Round 1 (Algorithm 8.6)
    Round 2 (Algorithm 8.7)
    Round 3 (Algorithm 8.8)
    Round 4 (Algorithm 8.9)
    $A \leftarrow A + A'$
    $B \leftarrow B + B'$
    $C \leftarrow C + C'$
    $D \leftarrow D + D'$
$(h(m) = A \parallel B \parallel C \parallel D)$

The complete descriptions of rounds 1, 2, and 3 of the MD4 hash algorithm are given in Algorithms 8.2–8.4. The constants c_1 and c_2 employed in rounds 2 and 3 refer to $c_1 = \lfloor 2^{30}\sqrt{2} \rfloor$ = `0x5A827999` and $c_2 = \lfloor 2^{30}\sqrt{3} \rfloor$ = `0x6ED9EBA1`.

A reference implementation of the MD4 hash algorithm (in the C programming language) is provided in Appendix A of RFC 1320 [4].

Algorithm 8.6 Round 1 of the MD5 hash function.

1. $A \leftarrow (A + f(B, C, D) + X[0] + T[1]) \hookleftarrow 7$
2. $D \leftarrow (D + f(A, B, C) + X[1] + T[2]) \hookleftarrow 12$
3. $C \leftarrow (C + f(D, A, B) + X[2] + T[3]) \hookleftarrow 17$
4. $B \leftarrow (B + f(C, D, A) + X[3] + T[4]) \hookleftarrow 22$
5. $A \leftarrow (A + f(B, C, D) + X[4] + T[5]) \hookleftarrow 7$
6. $D \leftarrow (D + f(A, B, C) + X[5] + T[6]) \hookleftarrow 12$
7. $C \leftarrow (C + f(D, A, B) + X[6] + T[7]) \hookleftarrow 17$
8. $B \leftarrow (B + f(C, D, A) + X[7] + T[8]) \hookleftarrow 22$
9. $A \leftarrow (A + f(B, C, D) + X[8] + T[9]) \hookleftarrow 7$
10. $D \leftarrow (D + f(A, B, C) + X[9] + T[10]) \hookleftarrow 12$
11. $C \leftarrow (C + f(D, A, B) + X[10] + T[11]) \hookleftarrow 17$
12. $B \leftarrow (B + f(C, D, A) + X[11] + T[12]) \hookleftarrow 22$
13. $A \leftarrow (A + f(B, C, D) + X[12] + T[13]) \hookleftarrow 7$
14. $D \leftarrow (D + f(A, B, C) + X[13] + T[14]) \hookleftarrow 12$
15. $C \leftarrow (C + f(D, A, B) + X[14] + T[15]) \hookleftarrow 17$
16. $B \leftarrow (B + f(C, D, A) + X[15] + T[16]) \hookleftarrow 22$

8.3.2 MD5

As mentioned earlier, MD5 is a strengthened version of MD4. It was proposed in 1991 and is specified in RFC 1321 [7]. There are only a few differences between MD4 and MD5, the most obvious being that MD5 uses four rounds (instead of three). This is advantageous from a security viewpoint. It is, however, also disadvantageous from a performance viewpoint. In fact, the additional round decreases the performance of the hash function about 30% (as compared to MD4).

The MD5 hash function is conceptually and structurally similar to MD4. In fact, the padding of the message m works exactly the same way. Again, there are some auxiliary functions. The selection function f and the function h are defined the same way as for MD4. The majority function g has changed from

$$g(X, Y, Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$$

to

$$g(X, Y, Z) = ((X \wedge Z) \vee (Y \wedge (\neg Z))$$

to make it less symmetric. In addition, there is a new function i that is defined as follows:

Algorithm 8.7 Round 2 of the MD5 hash function.

1. $A \leftarrow (A + g(B, C, D) + X[1] + T[17]) \hookleftarrow 5$
2. $D \leftarrow (D + g(A, B, C) + X[6] + T[18]) \hookleftarrow 9$
3. $C \leftarrow (C + g(D, A, B) + X[11] + T[19]) \hookleftarrow 14$
4. $B \leftarrow (B + g(C, D, A) + X[0] + T[20]) \hookleftarrow 20$
5. $A \leftarrow (A + g(B, C, D) + X[5] + T[21]) \hookleftarrow 5$
6. $D \leftarrow (D + g(A, B, C) + X[10] + T[22]) \hookleftarrow 9$
7. $C \leftarrow (C + g(D, A, B) + X[15] + T[23]) \hookleftarrow 14$
8. $B \leftarrow (B + g(C, D, A) + X[4] + T[24]) \hookleftarrow 20$
9. $A \leftarrow (A + g(B, C, D) + X[9] + T[25]) \hookleftarrow 5$
10. $D \leftarrow (D + g(A, B, C) + X[14] + T[26]) \hookleftarrow 9$
11. $C \leftarrow (C + g(D, A, B) + X[3] + T[27]) \hookleftarrow 14$
12. $B \leftarrow (B + g(C, D, A) + X[8] + T[28]) \hookleftarrow 20$
13. $A \leftarrow (A + g(B, C, D) + X[13] + T[29]) \hookleftarrow 5$
14. $D \leftarrow (D + g(A, B, C) + X[2] + T[30]) \hookleftarrow 9$
15. $C \leftarrow (C + g(D, A, B) + X[7] + T[31]) \hookleftarrow 14$
16. $B \leftarrow (B + g(C, D, A) + X[12] + T[32]) \hookleftarrow 20$

$$i(X, Y, Z) \quad = \quad Y \oplus (X \vee (\neg Z))$$

The truth table of the logical functions f, g, h, i is illustrated in Table 8.3. Furthermore, the MD5 hash function uses a 64-element table T constructed from the sine function. Let $T[i]$ be the i^{th} element of the table, then

$$T[i] = \lfloor 4,294,967,296 \cdot |\sin(i)| \rfloor$$

where i is in radians. Because $4,294,967,296$ is equal to 2^{32} and $|\sin(i)|$ is a number between 0 and 1, each element of T is an integer that can be represented in 32 bits. Consequently, the table T provides a "randomized" set of 32-bit patterns, which should eliminate any regularities in the input data. The elements of T as employed by the MD5 hash function are listed in Table 8.4.

The MD5 hash function is overviewed in Algorithm 8.5. It is structurally similar to the MD4 hash function. The four rounds of MD5 are specified in Algorithms 8.6–8.9.

Again, a reference implementation of the MD5 hash function (in the C programming language) is provided in Appendix A of the relevant RFC 1321 [7].

Algorithm 8.8 Round 3 of the MD5 hash function.

1. $A \leftarrow (A + h(B, C, D) + X[5] + T[33]) \hookleftarrow 4$
2. $D \leftarrow (D + h(A, B, C) + X[8] + T[34]) \hookleftarrow 11$
3. $C \leftarrow (C + h(D, A, B) + X[11] + T[35]) \hookleftarrow 16$
4. $B \leftarrow (B + h(C, D, A) + X[14] + T[36]) \hookleftarrow 23$
5. $A \leftarrow (A + h(B, C, D) + X[1] + T[37]) \hookleftarrow 4$
6. $D \leftarrow (D + h(A, B, C) + X[4] + T[38]) \hookleftarrow 11$
7. $C \leftarrow (C + h(D, A, B) + X[7] + T[39]) \hookleftarrow 16$
8. $B \leftarrow (B + h(C, D, A) + X[10] + T[40]) \hookleftarrow 23$
9. $A \leftarrow (A + h(B, C, D) + X[13] + T[41]) \hookleftarrow 4$
10. $D \leftarrow (D + h(A, B, C) + X[0] + T[42]) \hookleftarrow 11$
11. $C \leftarrow (C + h(D, A, B) + X[3] + T[43]) \hookleftarrow 16$
12. $B \leftarrow (B + h(C, D, A) + X[6] + T[44]) \hookleftarrow 23$
13. $A \leftarrow (A + h(B, C, D) + X[9] + T[45]) \hookleftarrow 4$
14. $D \leftarrow (D + h(A, B, C) + X[12] + T[46]) \hookleftarrow 11$
15. $C \leftarrow (C + h(D, A, B) + X[15] + T[47]) \hookleftarrow 16$
16. $B \leftarrow (B + h(C, D, A) + X[2] + T[48]) \hookleftarrow 23$

8.3.3 SHA-1

The SHA-1 hash function is conceptually and structurally similar to MD4 and MD5. The two most important differences are that SHA-1 was designed to run optimally on computer systems with a big-endian architecture (rather than a little-endian architecture) and that it employs five registers (instead of four) and hence outputs hash values of 160 bits.

The SHA-1 hash function uses a sequence of functions $f_0, f_1, \ldots, f_{79}$ that are defined as follows:

$$f_t(X, Y, Z) = \begin{cases} Ch(X, Y, Z) = (X \wedge Y) \oplus ((\neg X) \wedge Z) & 0 \leq t \leq 19 \\ Parity(X, Y, Z) = X \oplus Y \oplus Z & 20 \leq t \leq 39 \\ Maj(X, Y, Z) = (X \wedge Y) \oplus (X \wedge Z) \oplus (Y \wedge Z) & 40 \leq t \leq 59 \\ Parity(X, Y, Z) = X \oplus Y \oplus Z & 60 \leq t \leq 79 \end{cases}$$

The truth table of the logical functions employed by SHA-1 is illustrated in Table 8.5.

Furthermore, the function uses a sequence of 80 constant 32-bit words K_0, $K_1, \ldots, K_{79}$ that are defined as follows:

Algorithm 8.9 Round 4 of the MD5 hash function.

1. $A \leftarrow (A + i(B, C, D) + X[0] + T[49]) \hookleftarrow 6$
2. $D \leftarrow (D + i(A, B, C) + X[7] + T[50]) \hookleftarrow 10$
3. $C \leftarrow (C + i(D, A, B) + X[14] + T[51]) \hookleftarrow 15$
4. $B \leftarrow (B + i(C, D, A) + X[5] + T[52]) \hookleftarrow 21$
5. $A \leftarrow (A + i(B, C, D) + X[12] + T[53]) \hookleftarrow 6$
6. $D \leftarrow (D + i(A, B, C) + X[3] + T[54]) \hookleftarrow 10$
7. $C \leftarrow (C + i(D, A, B) + X[10] + T[55]) \hookleftarrow 15$
8. $B \leftarrow (B + i(C, D, A) + X[1] + T[56]) \hookleftarrow 21$
9. $A \leftarrow (A + i(B, C, D) + X[8] + T[57]) \hookleftarrow 6$
10. $D \leftarrow (D + i(A, B, C) + X[15] + T[58]) \hookleftarrow 10$
11. $C \leftarrow (C + i(D, A, B) + X[6] + T[59]) \hookleftarrow 15$
12. $B \leftarrow (B + i(C, D, A) + X[13] + T[60]) \hookleftarrow 21$
13. $A \leftarrow (A + i(B, C, D) + X[4] + T[61]) \hookleftarrow 6$
14. $D \leftarrow (D + i(A, B, C) + X[11] + T[62]) \hookleftarrow 10$
15. $C \leftarrow (C + i(D, A, B) + X[2] + T[63]) \hookleftarrow 15$
16. $B \leftarrow (B + i(C, D, A) + X[9] + T[64]) \hookleftarrow 21$

Table 8.3
Truth Table of the Logical Functions Employed by MD5

X	Y	Z	f	g	h	i
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

$$K_t = \begin{cases} \lfloor 2^{30}\sqrt{2} \rfloor = \texttt{0x5A827999} & 0 \leq t \leq 19 \\ \lfloor 2^{30}\sqrt{3} \rfloor = \texttt{0x6ED9EBA1} & 20 \leq t \leq 39 \\ \lfloor 2^{30}\sqrt{5} \rfloor = \texttt{0x8F1BBCDC} & 40 \leq t \leq 59 \\ \lfloor 2^{30}\sqrt{10} \rfloor = \texttt{0xCA62C1D6} & 60 \leq t \leq 79 \end{cases}$$

Note that the first two values correspond to c_1 and c_2 employed by MD4.

The preprocessing of the message to be hashed is identical to the one employed by MD4 and MD5. Because the SHA-1 hash function was designed to run on a big-endian architecture, the final two 32-bit words specifying the bit length s is appended with the most significant word preceding the least significant word.

Table 8.4
The Elements of Table T Employed by the MD5 Hash Function

$T[1]$=0xD76AA478	$T[17]$=0xF61E2562	$T[33]$=0xFFFA3942	$T[49]$=0xF4292244
$T[2]$=0xE8C7B756	$T[18]$=0xC040B340	$T[34]$=0x8771F681	$T[50]$=0x432AFF97
$T[3]$=0x242070DB	$T[19]$=0x265E5A51	$T[35]$=0x6D9D6122	$T[51]$=0xAB9423A7
$T[4]$=0xC1BDCEEE	$T[20]$=0xE9B6C7AA	$T[36]$=0xFDE5380C	$T[52]$=0xFC93A039
$T[5]$=0xF57C0FAF	$T[21]$=0xD62F105D	$T[37]$=0xA4BEEA44	$T[53]$=0x655B59C3
$T[6]$=0x4787C62A	$T[22]$=0x02441453	$T[38]$=0x4BDECFA9	$T[54]$=0x8F0CCC92
$T[7]$=0xA8304613	$T[23]$=0xD8A1E681	$T[39]$=0xF6BB4B60	$T[55]$=0xFFEFF47D
$T[8]$=0xFD469501	$T[24]$=0xE7D3FBC8	$T[40]$=0xBEBFBC70	$T[56]$=0x85845DD1
$T[9]$=0x698098D8	$T[25]$=0x21E1CDE6	$T[41]$=0x289B7EC6	$T[57]$=0x6FA87E4F
$T[10]$=0x8B44F7AF	$T[26]$=0xC33707D6	$T[42]$=0xEAA127FA	$T[58]$=0xFE2CE6E0
$T[11]$=0xFFFF5BB1	$T[27]$=0xF4D50D87	$T[43]$=0xD4EF3085	$T[59]$=0xA3014314
$T[12]$=0x895CD7BE	$T[28]$=0x455A14ED	$T[44]$=0x04881D05	$T[60]$=0x4E0811A1
$T[13]$=0x6B901122	$T[29]$=0xA9E3E905	$T[45]$=0xD9D4D039	$T[61]$=0xF7537E82
$T[14]$=0xFD987193	$T[30]$=0xFCEFA3F8	$T[46]$=0xE6DB99E5	$T[62]$=0xBD3AF235
$T[15]$=0xA679438E	$T[31]$=0x676F02D9	$T[47]$=0x1FA27CF8	$T[63]$=0x2AD7D2BB
$T[16]$=0x49B40821	$T[32]$=0x8D2A4C8A	$T[48]$=0xC4AC5665	$T[64]$=0xEB86D391

In addition to f_t and K_t, there is a message schedule W that comprises eighty 32-bit words. The schedule is initialized as follows:

$$W_t = \begin{cases} M[t] & 0 \leq t \leq 15 \\ (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \hookleftarrow 1 & 16 \leq t \leq 79 \end{cases}$$

After preprocessing is completed, the message is hashed iteratively using Algorithm 8.10. First, the five registers are initialized (the first four registers are indentically initialized as in the case of MD4, and the fifth register is initialized with 0xC3D2E1F0). Afterwards, each message block $M[1]$, $M[1]$, ..., $M[N]$ is processed iteratively, where the result of each iteration is used in the next iteration. Finally, the result is the concatenation of the values of the five registers. It is a 160-bit hash value that may serve as message digest for message m.

From a security viewpoint, there are two remarks to make at this point:

- First, a SHA-1 hash value is 32 bits longer than an MD5 hash value. This is advantageous, because it means that SHA-1 is potentially more resistant against brute-force attacks. This is even more true for the SHA variants itemized in Table 8.1.

Table 8.5
Truth Table of the Logical Functions Employed by SHA-1

X	Y	Z	$Ch = f_{0\ldots19}$	$Parity = f_{20\ldots39}$	$Maj = f_{40\ldots59}$	$Parity = f_{60\ldots79}$
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	0	1	0	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	1	0	0	1	0
1	1	0	1	0	1	0
1	1	1	1	1	1	1

- Second, SHA-1 appears not to be vulnerable to the attacks against MD4 and MD5. However, little is publicly known about the design criteria for SHA-1, so its strength is somehow difficult to evaluate.

On the other hand, SHA-1 involves more steps (80 as compared to 64) and must process a 160-bit register compared to the 128-bit register of MD4 and MD5. Consequently, SHA-1 executes a little bit more slowly.

8.4 FINAL REMARKS

In this chapter, we elaborated on cryptographic hash functions. Most of these functions that are practically relevant (e.g., MD5 and SHA-1) follow the Merkle-Damgård construction. This also applies to some more recent alternatives, such as Whirlpool.[17] The fact that cryptographic hash function follows the Merkle-Damgård construction basically means that a collision resistant compression function is iterated multiple times (one iteration for each block of the message). Each iteration can only start if the preceding iteration has finished. This suggests that the resulting cryptographic hash function may become a performance bottleneck. For example, Joe Touch showed that the currently achievable hash rates of MD5 are insufficient to keep up with high-speed networks [17]. The problem is the iterative nature of MD5 and its block chaining structure, which prevent parallelism. As also shown in [17], it is possible to modify the MD5 algorithm to accommodate a slightly higher throughput. Alternatively, it is possible to design and come up with cryptographic hash functions that are inherently more qualified to provide support for parallelism.

Although most cryptographic hash functions in use today follow the Merkle-Damgård construction, the design of the underlying compression functions still

17 http://planeta.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html

Algorithm 8.10 The SHA-1 hash function (overview).

$(m = m_0 m_1 \dots m_{s-1})$
Construct $M = M[0]M[1] \dots M[N-1]$
$A \leftarrow$ `0x67452301`
$B \leftarrow$ `0xEFCDAB89`
$C \leftarrow$ `0x98BADCFE`
$D \leftarrow$ `0x10325476`
$E \leftarrow$ `0xC3D2E1F0`
for $i = 0$ to N do
    Prepare the message schedule W
    $A' \leftarrow A$
    $B' \leftarrow B$
    $C' \leftarrow C$
    $D' \leftarrow D$
    $E' \leftarrow E$
    for $t = 0$ to 79 do
        $T \leftarrow (A \hookleftarrow 5) + f_t(B, C, D) + E + K_t + W_t$
        $E \leftarrow D$
        $D \leftarrow C$
        $C \leftarrow B \hookleftarrow 30$
        $B \leftarrow A$
        $A \leftarrow T$
    $A \leftarrow A + A'$
    $B \leftarrow B + B'$
    $C \leftarrow C + C'$
    $D \leftarrow D + D'$
    $E \leftarrow E + E'$
$(h(m) = A \parallel B \parallel C \parallel D \parallel E)$

looks more like an art than a science. For example, finding collisions in such functions has recently become a very active area of research (e.g., [18]). Remember from Section 8.3 that collisions were recently found for MD4, MD5, and some other cryptographic hash functions. Also, as this book went to press, a group of Chinese researchers claimed to have found an attack that requires only 2^{69} (instead of 2^{80}) hash operations to find a collision in SHA-1.[18]

As of this writing, there are hardly any design criteria that can be used to design and come up with new compression functions (for cryptographic hash functions that follow the Merkle-Damgård construction) or entirely new cryptographic hash functions. This lack of design criteria is somehow in contrast to the relative importance of cryptographic hash functions in almost all cryptographic systems and applications. Consequently, an interesting and challenging area of research and

18 http://theory.csail.mit.edu/∼yiqun/shanote.pdf

development would try to specify design criteria for compression functions (if the Merkle-Damgård construction is used) or entirely new cryptographic hash functions (if the Merkle-Damgård construction is not used). For example, *universal hashing* as originally proposed in the late 1970s by Larry Carter and Mark Wegman [19, 20] provides an interesting design paradigm for new cryptographic hash functions. Instead of using a single hash function, universal hashing considers families of hash functions. The hash function in use is then chosen randomly from the family. We briefly revisit the topic when we address MACs using families of universal hash functions in Section 11.2.4.

References

[1] Merkle, R.C., "One Way Hash Functions and DES," *Proceedings of CRYPTO '89*, Springer-Verlag, LNCS 435, 1989, pp. 428–446.

[2] Damgård, I.B., "A Design Principle for Hash Functions," *Proceedings of CRYPTO '89*, Springer-Verlag, LNCS 435, 1989, pp. 416–427.

[3] Kaliski, B., *The MD2 Message-Digest Algorithm*, Request for Comments 1319, April 1992.

[4] Rivest, R.L., *The MD4 Message-Digest Algorithm*, Request for Comments 1320, April 1992.

[5] Biham, E., and A. Shamir, "Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI, and Lucifer," *Proceedings of CRYPTO '91*, Springer-Verlag, LNCS 576, 1991, pp. 156–171.

[6] den Boer, B., and A. Bosselaers, "An Attack on the Last Two Rounds of MD4," *Proceedings of CRYPTO '91*, Springer-Verlag, LNCS 576, 1991, pp. 194–203.

[7] Rivest, R.L., *The MD5 Message-Digest Algorithm*, Request for Comments 1321, April 1992.

[8] Dobbertin, H., "Cryptanalysis of MD4," *Journal of Cryptology*, Vol. 11, No. 4, 1998, pp. 253–271.

[9] Dobbertin, H., "The Status of MD5 After a Recent Attack," *CryptoBytes*, Vol. 2, No. 2, Summer 1996.

[10] den Boer, B., and A. Bosselaers, "Collisions for the Compression Function of MD5," *Proceedings of EUROCRYPT '93*, Springer-Verlag, LNCS 765, 1993, pp. 293–304.

[11] Chabaud, F., and A. Joux, "Differential Collisions in SHA-0," *Proceedings of CRYPTO '98*, Springer-Verlag, LNCS 1462, 1998, pp. 56–71.

[12] U.S. Department of Commerce, National Institute of Standards and Technology, *Secure Hash Standard*, FIPS PUB 180-1, April 1995.

[13] Eastlake, D., and P. Jones, *US Secure Hash Algorithm 1 (SHA1)*, Request for Comments 3174, September 2001.

[14] Housley, R., *A 224-Bit One-Way Hash Function: SHA-224*, Request for Comments 3874, September 2004.

[15] Dobbertin, H., A. Bosselaers, and B. Preneel, "RIPEMD-160: A Strengthened Version of RIPEMD," *Proceedings of the 3rd International Workshop on Fast Software Encryption*, Springer-Verlag, LNCS 1039, 1996, pp. 71–82.

[16] Preneel, B., A. Bosselaers, and H. Dobbertin, "The Cryptographic Hash Function RIPEMD-160," *CryptoBytes*, Vol. 3, No. 2, 1997, pp. 9–14.

[17] Touch, J., *Report on MD5 Performance*, Request for Comments 1810, June 1995.

[18] Biham, E., and R. Chen, "Near-Collisions of SHA-0," *Proceedings of CRYPTO 2004*, Springer-Verlag, LNCS 3152, 2004.

[19] Carter, J.L., and M.N. Wegman, "Universal Classes of Hash Functions," *Journal of Computer and System Sciences*, Vol. 18, 1979, pp. 143–154.

[20] Carter, J.L., and M.N. Wegman, "New Hash Functions and Their Use in Authentication and Set Equality," *Journal of Computer and System Sciences*, Vol. 22, 1981, pp. 265–279.