CHAPTER 4

CRYPTOGRAPHIC HASH FUNCTIONS AND DIGITAL SIGNATURES

CHAPTER OUTLINE

After studying this chapter, the students will be able to understand the

- Message Authentication, Message Authentication Functions, Message Authentication Codes
- Hash Functions, Properties of Hash functions, Applications of Hash Functions
- Message Digests: MD4 and MD5
- Secure Hash Algorithms: SHA-1 and SHA-2
- Digital Signatures: Direct Digital Signatures, Arbitrated Digital Signature
- Digital Signature Standard: The DSS Approach, Digital Signature Algorithm
- Digital Signature Standard: The RSA Approach

MESSAGE AUTHENTICATION

Encryption protects against passive attack (eavesdropping). A different requirement is to protect against active attack (falsification of data and transactions). Protection against such attacks is known as message authentication.

A message, file, document, or other collection of data is said to be authentic when it is genuine and comes from its alleged source. Message authentication is a procedure that allows communicating parties to verify that received messages are authentic. The two important aspects are to verify that the contents of the message have not been altered and that the source is authentic. We may also wish to verify a message's timeliness (it has not been artificially delayed and replayed) and sequence relative to other messages flowing between two parties. All of these concerns come under the category of data integrity.

Message authentication is a mechanism or service used to verify the integrity of a message. Message authentication assures that data received are exactly as sent by (i.e., contain no modification, insertion, deletion, or replay) and that the purported identity of the sender is valid.

In the context of communications across a network, the following attacks can be identified.

- **Disclosure:** Release of message contents to any person or process not possessing the appropriate cryptographic key.
- **Traffic analysis:** Discovery of the pattern of traffic between parties. In a connection-oriented application, the frequency and duration of connections could be determined. In either a connection-oriented or connectionless environment, the number and length of messages between parties could be determined.
- **Masquerade:** Insertion of messages into the network from a fraudulent source. This includes the creation of messages by an opponent that are purported to come from an authorized entity. Also included are fraudulent acknowledgments of message receipt or nonreceipt by someone other than the message recipient.
- **Content modification:** Changes to the contents of a message, including insertion, deletion, transposition, and modification.
- **Sequence modification:** Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.
- **Timing modification:** Delay or replay of messages. In a connection-oriented application, an entire session or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed. In a connectionless application, an individual message (e.g., datagram) could be delayed or replayed.
- **Source repudiation:** Denial of transmission of message by source.
- **Destination repudiation:** Denial of receipt of message by destination.

In summary, message authentication is a procedure to verify that received messages come from the alleged source and have not been altered. Message authentication may also verify sequencing and timeliness. A digital signature is an authentication technique that also includes measures to counter repudiation by the source.

MESSAGE AUTHENTICATION FUNCTIONS

Any message authentication or digital signature mechanism has two levels of functionality. At the lower level, there must be some sort of function that produces an authenticator: a value to be used to authenticate a message. This lower-level function is then used as a primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of a message.

The types of functions that may be used to produce an authenticator may be grouped into three classes:

- **Hash function:** A function that maps a message of any length into a fixed-length hash value, which serves as the authenticator
- **Message encryption:** The ciphertext of the entire message serves as its authenticator
- **Message authentication code (MAC):** A function of the message and a secret key that produces a fixed-length value that serves as the authenticator

MESSAGE AUTHENTICATION CODES

One authentication technique involves the use of a secret key to generate a small block of data, known as a **message authentication code (MAC)** that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key *K*. When A has a message to send to B, it calculates the message authentication code as a function of the message and the key:

$$MAC = F(K, M)$$

Where,

M = input message

F = MAC function

K = shared secret key

MAC = message authentication code

The message plus code are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the same secret key, to generate a new message authentication code. The received code is compared to the calculated code (Figure 4.1). If we assume that only the receiver and the sender know the identity of the secret key, and if the received code matches the calculated code, then the following statements apply:

1. The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the code, then the receiver's calculation of the code will differ from the received code. Because the attacker is assumed not to know the secret key, the attacker cannot alter the code to correspond to the alterations in the message.

2. The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key, no one else could prepare a message with a proper code.
3. If the message includes a sequence number (such as is used with HDLC and TCP), then the receiver can be assured of the proper sequence, because an attacker cannot successfully alter the sequence number.

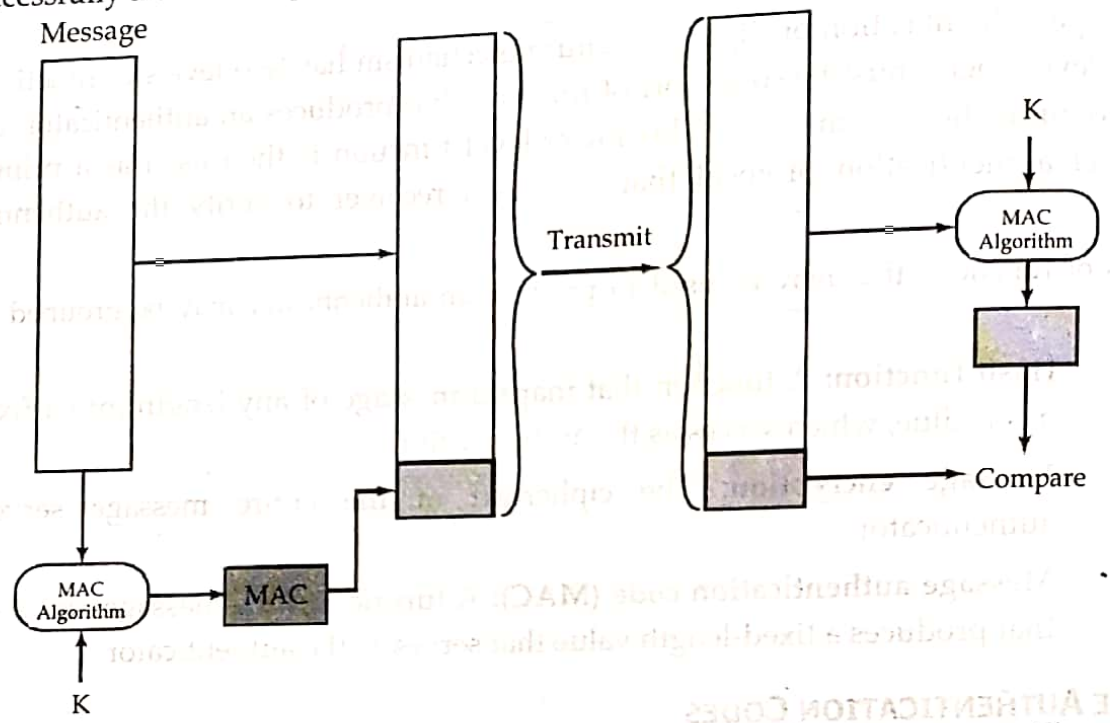


Figure: Message Authentication Using a Message Authentication Code (MAC)

The process just described is similar to encryption. One difference is that the authentication algorithm need not be reversible, as it must for decryption. Because of the mathematical properties of the authentication function, it is less vulnerable to being broken than encryption.

CRYPTOGRAPHIC HASH FUNCTIONS

Cryptographic hash functions are an important tool of cryptography and play a fundamental role in efficient and secure information processing. A hash function processes an arbitrary finite length input message to a fixed length output referred to as the hash value. As a security requirement, a hash value should not serve as an image for two distinct input messages and it should be difficult to find the input message from a given hash value. Secure hash functions serve data integrity, non-repudiation and authenticity of the source in conjunction with the digital signature schemes. Keyed hash functions, also called message authentication codes (MACs) serve data integrity and data origin authentication in the secret key setting. The building blocks of hash functions can be designed using block ciphers, modular arithmetic or from scratch. The design principles of the popular Merkle–Damgård construction are followed in almost all widely used standard hash functions such as MD5 and SHA-1.

In the last few years, collision attacks on the MD5 and SHA-1 hash functions have been demonstrated and weaknesses in the Merkle–Damgård construction have been exposed. The

impact of these attacks on some important applications has also been analysed. This successful cryptanalysis of the standard hash functions has made National Institute of Standards and Technology (NIST), USA to initiate an international public competition to select the most secure and efficient hash function as the Advanced Hash Standard (AHS) which will be referred to as SHA-3.

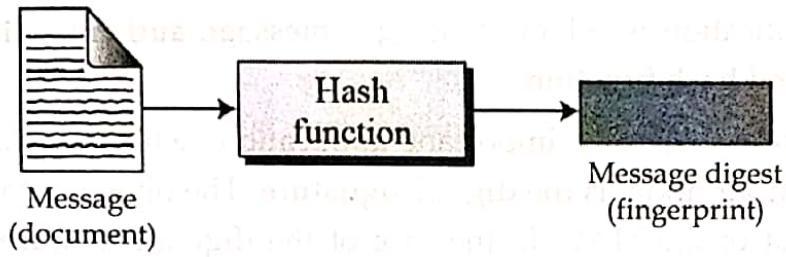


Figure 4.2: Message and digest

PROPERTIES OF CRYPTOGRAPHIC HASH FUNCTIONS

The hash computation on a message m is mathematically represented by $H(m) = y$. The computation of y from m must be easy and fast. The fundamental security properties of H are defined below (see figure 4.3):

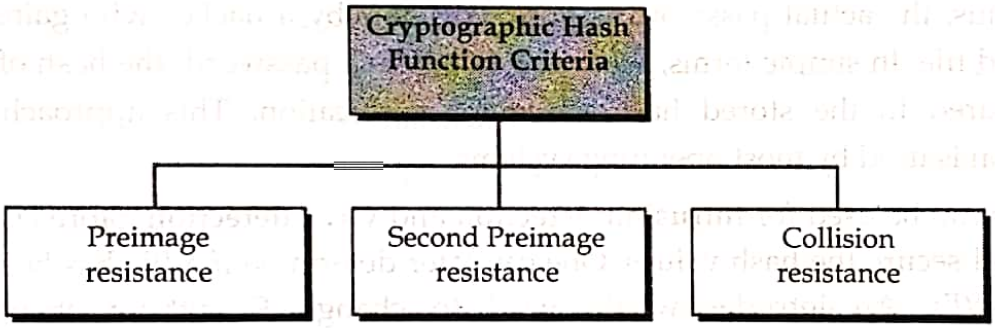


Figure 4.3: Criteria of a cryptographic hash functions

- **Preimage resistance:** H is preimage resistant if for any given hash value y of H, it is "computationally infeasible" to find a message m such that $H(m) = y$. That is, it must be hard to invert H from y to get an m corresponding to y. This property is also called one-wayness. For an ideal H, it takes about 2^n evaluations of H to find a preimage.
- **Second pre-image resistance:** H is second preimage resistant if for any given message m, it is "computationally infeasible" to find another message m^* such that $m^* = m$ and $H(m) = H(m^*)$. For an ideal H, it takes about 2^n evaluations of H to find a second preimage.
- **Collision resistance:** H is collision resistant if it is "computationally infeasible" to find any two messages m and m^* such that $m \neq m^*$ and $H(m) = H(m^*)$. Due to the birthday paradox, for an ideal H, it takes about $2^{n/2}$ evaluations of H to find a collision.

APPLICATIONS OF CRYPTOGRAPHIC HASH FUNCTIONS

Perhaps the most versatile cryptographic algorithm is the cryptographic hash function. It is used in a wide variety of security applications and Internet protocols. To better understand some of the requirements and security implications for cryptographic hash functions, it is useful to look at the range of applications in which it is employed.

- **Message Authentication:** Message authentication is a mechanism or service used to verify the integrity of a message. Message authentication assures that data received are exactly as sent (i.e., contain no modification, insertion, deletion, or replay). In many cases, there is a requirement that the authentication mechanism assures that purported identity of the sender is valid. When a hash function is used to provide message authentication, the hash function value is often referred to as a **message digest.** More commonly, message authentication is achieved using a **message authentication code (MAC),** also known as a **keyed hash function.**
- **Digital Signatures:** Another important application, which is similar to the message authentication application, is the **digital signature**. The operation of the digital signature is similar to that of the MAC. In the case of the digital signature, the hash value of a message is encrypted with a user's private key. Anyone who knows the user's public key can verify the integrity of the message that is associated with the digital signature. In this case, an attacker who wishes to alter the message would need to know the user's private key.
- **Other Applications:** Hash functions are commonly used to create a **one-way password file**. Hash value of a password is stored by an operating system rather than the password itself. Thus, the actual password is not retrievable by a hacker who gains access to the password file. In simple terms, when a user enters a password, the hash of the password is compared to the stored hash value for verification. This approach to password protection is used by most operating systems.

Hash functions can be used for **intrusion detection** and **virus detection**. Store *H(F)* for each file on a system and secure the hash values. One can later determine if a file has been modified by recomputing *H(F)*. An intruder would need to change *F* without changing *H(F)*. A cryptographic hash function can be used to construct a **pseudorandom function (PRF)** or a **pseudorandom number generator (PRNG)**. A common application for a hash-based **PRF** is for the generation of symmetric keys.

MESSAGE DIGESTS

Professor Ronald Rivest is a professor in MIT who invented RSA, RC5 and the Message Digest (MD) hashing functions. Rivest first designed MD2 for 8-bit machines in 1989. The original message is padded at first so that the total message is divisible by 16. Plus a 16-byte checksum is added to it to create a total 128-bit message digest or hash value. But collisions were for MD2 were found soon. Rivest then developed MD4 for 32-bit machines in 1990. MD4 influenced a lot of cryptographic hash functions such as MD5, SHA-1. Same as MD2 collisions for MD4 were found soon enough. MD4 has been criticized even by Ronald Rivest because MD4 was designed to be fast which led to a lot of security risks. MD5 was developed in 1991 by Rivest. MD5 is almost same as MD4 but with "safety belts". It is slower than MD4 but more secure. But over the years collisions were found in MD5. Den Boer and Bosselaers first found collision in MD5 in 1993. In March 2004 a project called MD5CRK was initiated to find collision in MD5 by using Birthday Attack.

MD4 (MESSAGE DIGEST 4)

MD4 Algorithm is a cryptographic hash function developed by Ronald Rivest in 1990. It produces 128 bits (four 32 bits words) message digest and MD4 was designed to be 32-bit-word-oriented so that it can be computed faster on 32-bit CPUs than an octet-oriented scheme

OPERATIONS

- **MD4 Message Padding:**

The message to be processed by MD4 computation must be multiple of 512 bits (16 32-bit words). The original message is padded by adding a 1 bit, followed by enough 0 bits to leave the message 64 bits less than a multiple of 512 bits. Then a 64-bit quantity representing the number of bits in the unpadded message, mod 264, is appended to the message. The bit order within octets is most significant to least significant; the octet order is least significant to most significant.

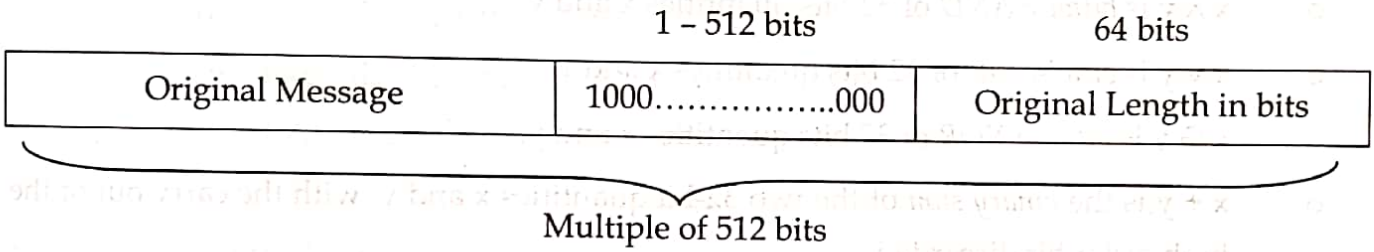


Figure 4.4: Padding of MD4

- **Overview of MD4 Message Digest Computation:**

The message digest to be computed is a 128-bit quantity (four 32-bit words). The message is processed in 512-bit (sixteen 32-bit words) blocks. (See figure 4.5.) The message digest is initialized to a fixed value, and then each stage of the message digest computation takes the current value of the message digest and modifies it using the next block of the message. The function that takes 512 bits of the message and digests it with the previous 128-bit output is known as the compression function. The final result is the message digest for the entire message.

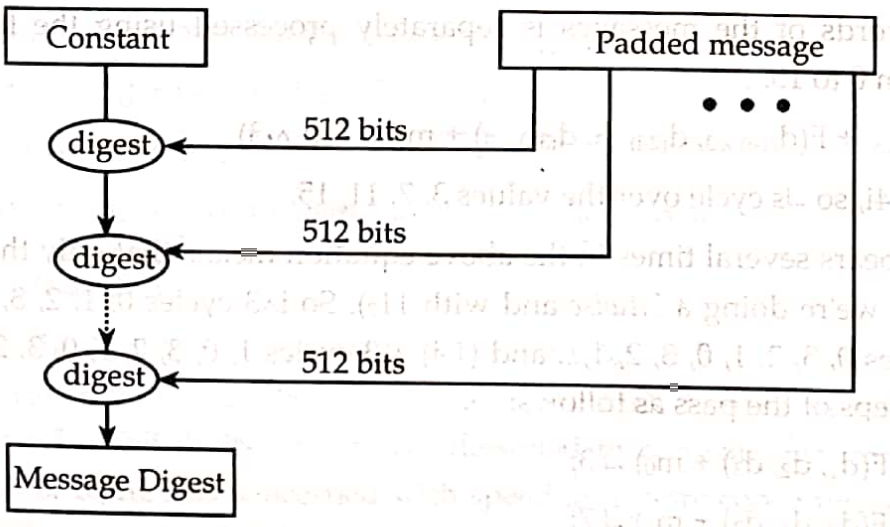


Figure 4.5: Overview of MD4, MD5, SHA-1

Each stage makes three passes over the message block. Each pass has a slightly different method of mangling the message digest. At the end of the stage, each word of the mangled message digest is added to its pre-stage value to produce the post-stage value (which becomes the pre-stage value for the next stage). Therefore, the current value of the message digest must be saved at the beginning of the stage so that it can be added in at the end of the stage.

Each stage starts with a 16-word message block and a 4-word message digest value. The message words are called $m_0, m_1, m_2, \ldots, m_{15}$. The message digest words are called d_0, d_1, d_2, d_3. Before the first stage the message digest is initialized to $d_0 = 67452301_{16}$, $d_1 = efcdab89_{16}$, $d_2 = 98badcfe_{16}$, $d_3 = 10325476_{16}$, equivalent to the octet string (written as a concatenation of hex-encoded octets) 01 | 23 | 45 | 67 | 89 | ab | cd | ef | fe | dc | ba | 98 | 76 | 54 | 32 | 10.

Each pass modifies d_0, d_1, d_2, d_3 using $m_0, m_1, \ldots m_{15}$ with the following operations.

- $\lfloor x \rfloor$ is the *floor* of the number x, i.e., the greatest integer not greater than x.
- $\sim x$ is *bitwise complement* of 32 bits quantity x.
- $x \wedge y$ is *bitwise AND* of 32 bits quantities x and y.
- $x \vee y$ is *bitwise OR* of 32 bits quantities x and y.
- $x \oplus y$ is *bitwise XOR* of 32 bits quantities x and y.
- $x + y$ is the *binary sum* of the two 32-bit quantities x and y, with the carry out of the high order bit discarded.
- $x \lrcorner y$ is the 32-bit quantity produced by taking the 32 bits of x and shifting them one position left y times, each time taking the bit shifted off the left end and placing it as the rightmost bit. This operation is known as a left rotate.

- **MD4 Message Digest Pass 1:**

A function $F(x, y, z) = (x \wedge y) \vee (\sim x \wedge z)$; takes three 32-bit words x, y, and z, and produces an output 32-bit word. This function is sometimes known as the **selection function**, because if the n^{th} bit of x is a 1 it selects the n^{th} bit of y for the n^{th} bit of the output. Otherwise (if the n^{th} bit of x is a 0) it selects the n^{th} bit of z for the n^{th} bit of the output.

Each of the 16 words of the messages is separately processed using the following relation, where i move from 0 to 15.

$$d_{(-i) \wedge 3} = (d_{(-i) \wedge 3} + F(d_{(1-i) \wedge 3}, d_{(2-i) \wedge 3}, d_{(3-i) \wedge 3}) + m_i) \lrcorner S_1(i \wedge 3)$$

Where, $S_1(i) = 3 + 4i$, so $\lrcorner$s cycle over the values 3, 7, 11, 15.

The "$\wedge 3$" that appears several times in the above equation means that only the bottom two bits are used (because we're doing a *bitwise* and with 11_2). So $i \wedge 3$ cycles 0, 1, 2, 3, 0, 1, 2, 3, ... while $(-i) \wedge 3$ cycles 0, 3, 2, 1, 0, 3, 2, 1,... and $(1-i) \wedge 3$ cycles 1, 0, 3, 2, 1, 0, 3, 2,.... We can write out the first few steps of the pass as follows:

$$d_0 = (d_0 + F(d_1, d_2, d_3) + m_0) \lrcorner 3;$$

$$d_3 = (d_3 + F(d_0, d_1, d_2) + m_1) \lrcorner 7;$$

$d_2 = (d_1 + F(d_3, d_0, d_1) + m_2) \lrcorner 11;$

$d_1 = (d_1 + F(d_2, d_3, d_1) + m_3) \lrcorner 15;$

$d_0 = (d_0 + F(d_1, d_2, d_3) + m_4) \lrcorner 3;$ and so on.

- **MD4 Message Digest Pass 2:**

A function $G(x, y, z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$ is sometimes known as the **majority function**, because the n^{th} bit of the output is a 1 iff at least two of the three input words' n^{th} bits are a 1. In this pass we introduce one strange constant $\lfloor 2^{30}\sqrt{2} \rfloor = 5a827999_{16}$. A separate step is done for each of the 16 words of the message. For each integer i from 0 through 15,

$$d_{(-i) \wedge 3} = (d_{(-i) \wedge 3} + G(d_{(1-i) \wedge 3}, d_{(2-i) \wedge 3}, d_{(3-i) \wedge 3}) + m_{X(i)} + 5a827999_{16}) \lrcorner S_2(i \wedge 3),$$

Where, X(i) is 4-bits number formed by exchanging the low order and high order pairs of bits in 4-bits number i (so, $X(i) = 4i - 15\lfloor i/4 \rfloor$), and $S_2(0) = 3$, $S_2(1) = 5$, $S_2(2) = 9$, $S_2(3) = 13$, so $\lrcorner s$ cycle over the values 3, 5, 9, 13. We can write out the first few steps of the pass as follows:

$d_0 = (d_0 + G(d_1, d_2, d_3) + m_0 + 5a827999_{16}) \lrcorner 3;$

$d_3 = (d_3 + G(d_0, d_1, d_2) + m_4 + 5a827999_{16}) \lrcorner 5;$

$d_2 = (d_1 + G(d_3, d_0, d_1) + m_8 + 5a827999_{16}) \lrcorner 9;$

$d_1 = (d_1 + G(d_2, d_3, d_1) + m_{12} + 5a827999_{16}) \lrcorner 13;$

$d_0 = (d_0 + G(d_1, d_2, d_3) + m_1 + 5a827999_{16}) \lrcorner 3;$ and so on.

- **MD4 Message Digest Pass 3:**

A function $H(x, y, z) = x \oplus y \oplus z$ that takes three 32 bits words gives a 32 bits output. In this pass a different strange constant $\lfloor 2^{30}\sqrt{3} \rfloor = 6ed9eba1_{16}$. A separate step is done for each of the 16 words of the message. For each integer i from 0 through 15,

$$d_{(-i) \wedge 3} = (d_{(-i) \wedge 3} + H(d_{(1-i) \wedge 3}, d_{(2-i) \wedge 3}, d_{(3-i) \wedge 3}) + m_{R(i)} + 6ed9eba1_{16}) \lrcorner S_3(i \wedge 3),$$

Where R(i) is 4-bits number formed by reversing the order of bits in 4-bits number i (so, $R(i) = 8i - 12\lfloor i/2 \rfloor - 6\lfloor i/4 \rfloor - 3\lfloor i/8 \rfloor$), and $S_3(0) = 3$, $S_3(1) = 9$, $S_3(2) = 11$, $S_3(3) = 15$, so $\lrcorner s$ cycle over the values 3, 9, 11, 15. We can write out the first few steps of the pass as follows:

$d_0 = (d_0 + H(d_1, d_2, d_3) + m_0 + 6ed9eba1_{16}) \lrcorner 3;$

$d_3 = (d_3 + H(d_0, d_1, d_2) + m_8 + 6ed9eba1_{16}) \lrcorner 9;$

$d_2 = (d_1 + H(d_3, d_0, d_1) + m_4 + 6ed9eba1_{16}) \lrcorner 11;$

$d_1 = (d_1 + H(d_2, d_3, d_1) + m_{12} + 6ed9eba1_{16}) \lrcorner 15;$

$d_0 = (d_0 + H(d_1, d_2, d_3) + m_2 + 6ed9eba1_{16}) \lrcorner 3;$ and so on.

MD 5 (MESSAGE DIGEST 5)

MD5 was designed by Ron Rivest in 1991 to replace an earlier hash function, MD4. MD5 is a widely used hash function with a 128-bit hash value. An MD5 hash is typically expressed as a sequence of 32 hexadecimal digits. MD5 was designed to be somewhat more "conservative" than MD4 in terms of being less concerned with speed and more concerned with security. It is very similar to MD4. The major differences are:

1. MD4 makes three passes over each 16-octet chunk of the message. MD5 makes four passes over each 16-octet chunk.
2. The functions are slightly different, as are the number of bits in the shifts.
3. MD4 has one constant which is used for each message word in pass 2, and a different constant used for the entire 16 message words in pass 3. No constant is used in pass 1.

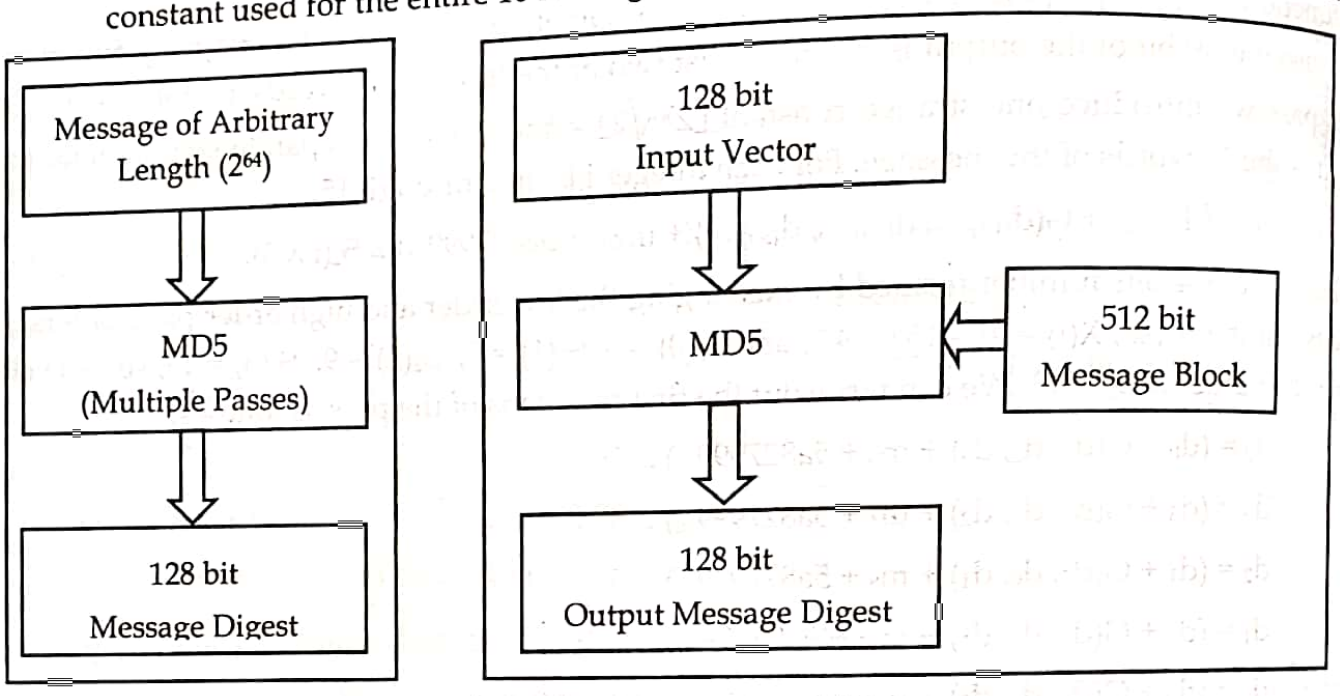


Figure 4.6: Basic Model of MD5

CONSTANTS USED IN MD 5

MD5 uses a different constant for each message word on each pass. Since there are 4 passes, each of which deals with 16 message words, there are 64 32-bit constants used in MD5. We will call them T_1 through T_{64}. T_i is based on the sine function i.e. $T_i = \lfloor 2^{32} | \sin i | \rfloor$ and the 64 values (in hex) are:

T_1 = d76aa478	T_{17} = f61e2562	T_{33} = fffa3942	T_{49} = f4292244
T_2 = e8c7b756	T_{18} = c040b340	T_{34} = 8771f681	T_{50} = 432aff97
T_3 = 242070db	T_{19} = 265e5a51	T_{35} = 6d9d6122	T_{51} = ab9423a7
T_4 = c1bdceee	T_{20} = e9b6c7aa	T_{36} = fde5380c	T_{52} = fc93a039
T_5 = f57c0faf	T_{21} = d62f105d	T_{37} = a4beea44	T_{53} = 655b59c3
T_6 = 4787c62a	T_{22} = 2441453	T_{38} = 4bdecfa9	T_{54} = 8f0ccc92
T_7 = a8304613	T_{23} = d8a1e681	T_{39} = f6bb4b60	T_{55} = ffeff47d
T_8 = fd469501	T_{24} = e7d3fbc8	T_{40} = bebfbc70	T_{56} = 85845dd1
T_9 = 698098d8	T_{25} = 21e1cde6	T_{41} = 289b7ec6	T_{57} = 6fa87e4f
T_{10} = 8b44f7af	T_{26} = c33707d6	T_{42} = eaa127fa	T_{58} = fe2ce6e0
T_{11} = ffff5bb1	T_{27} = f4d50d87	T_{43} = d4ef3085	T_{59} = a3014314
T_{12} = 895cd7be	T_{28} = 455a14ed	T_{44} = 4881d05	T_{60} = 4e0811a1
T_{13} = 6b901122	T_{29} = a9e3e905	T_{45} = d9d4d039	T_{61} = f7537e82
T_{14} = fd987193	T_{30} = fcefa3f8	T_{46} = e6db99e5	T_{62} = bd3af235
T_{15} = a679438e	T_{31} = 676f02d9	T_{47} = 1fa27cf8	T_{63} = 2ad7d2bb
T_{16} = 49b40821	T_{32} = 8d2a4c8a	T_{48} = c4ac5665	T_{64} = eb86d391

OPERATIONS OF MD5

- **MD5 Message Padding:**

The padding in MD5 is identical to the padding in MD4.

- **Overview of MD5 Message Digest Computation:**

Like MD4, in MD5 the message is processed in 512-bit blocks (sixteen 32-bit words). (See figure 4.5.) The message digest is a 128-bit quantity (four 32-bit words). Each stage consists of computing a function based on the 512-bit message chunk and the message digest to produce a new intermediate value for the message digest. The value of the message digest is the result of the output of the final block of the message.

Each stage in MD5 takes four passes over the message block (as opposed to three for MD4). As with MD4, at the end of the stage, each word of the modified message digest is added to the corresponding pre-stage message digest value. And as in MD4, before the first stage the message digest is initialized to $d_0 = 67452301_{16}$, $d_1 = efcdab89_{16}$, $d_2 = 98badcfe_{16}$, and $d_3 = 10325476_{16}$.

As with MD4, each pass modifies d_0, d_1, d_2, d_3 using m_0, m_1, m_2,...m_{15}. Each pass are described separately as following:

- **MD5 Message Digest Pass 1:**

As with MD4 this pass uses the function $F(x, y, z) = (x \wedge y) \vee (\sim x \wedge z)$. A separate step is done for each of the 16 words of the message. For each integer i from 0 through 15,

$$d_{(-i)\wedge 3} = d_{(1-i)\wedge 3} + (d_{(-i)\wedge 3} + F(d_{(1-i)\wedge 3}, d_{(2-i)\wedge 3}, d_{(3-i)\wedge 3}) + m_i + T_i) \lrcorner S_1(i \wedge 3),$$

Where, $S_1(i) = 7 + 5i$ so $\lrcorner$s cycle over the values 7, 12, 17, 22. This is a different S_1 from that in MD4. We can write out the first few steps of the pass as follows:

$$d_0 = d_1 + (d_0 + F(d_1, d_2, d_3) + m_0 + T_1) \lrcorner 7;$$

$$d_3 = d_0 + (d_3 + F(d_0, d_1, d_2) + m_1 + T_2) \lrcorner 12;$$

$$d_2 = d_3 + (d_1 + F(d_3, d_0, d_1) + m_2 + T_3) \lrcorner 17;$$

$$d_1 = d_2 + (d_1 + F(d_2, d_3, d_1) + m_3 + T_4) \lrcorner 22;$$

$$d_0 = d_1 + (d_0 + F(d_1, d_2, d_3) + m_4 + T_5) \lrcorner 7;$$ and so on.

- **MD5 Message Digest Pass 2:**

This pass uses the function $G(x, y, z) = (x \wedge z) \vee (y \wedge \sim z)$. Whereas the function F was the same in MD5 as in MD4, the function G is different in MD5 than the G function in MD4. In fact, MD5's G is rather like F the n^{th} bit of z is used to select the n^{th} bit in x or the n^{th} bit in y. A separate step is done for each of the 16 words of the message. For each integer i from 0 through 15,

$$d_{(-i)\wedge 3} = d_{(1-i)\wedge 3} + (d_{(-i)\wedge 3} + G(d_{(1-i)\wedge 3}, d_{(2-i)\wedge 3}, d_{(3-i)\wedge 3}) + m_{(5i+1)\wedge 15} + T_{i+17}) \lrcorner S_2(i \wedge 3),$$

Where $S_2(i) = i(i+7)/2 + 5$ so ↵s cycles over the values 5, 9, 14, 20. This is a different S2 from that in MD4. We can write out the first few steps of the pass as follows:

$d_0 = d_1 + (d_0 + G(d_1, d_2, d_3) + m_1 + T_{17}) \lhd 5;$

$d_3 = d_0 + (d_3 + G(d_0, d_1, d_2) + m_6 + T_{18}) \lhd 9;$

$d_2 = d_3 + (d_1 + G(d_3, d_0, d_1) + m_{11} + T_{19}) \lhd 14;$

$d_1 = d_2 + (d_1 + G(d_2, d_3, d_1) + m_0 + T_{20}) \lhd 20;$

$d_0 = d_1 + (d_0 + G(d_1, d_2, d_3) + m_5 + T_{21}) \lhd 5;$ and so on.

- **MD5 Message Digest Pass 3:**

This pass uses the function $H(x, y, z) = x \oplus y \oplus z$. This is the same H as in MD4. A separate step is done for each of the 16 words of the message. For each integer i from 0 through 15,

$$d_{(-i) \wedge 3} = d_{(1-i) \wedge 3} + (d_{(-i) \wedge 3} + H(d_{(1-i) \wedge 3}, d_{(2-i) \wedge 3}, d_{(3-i) \wedge 3}) + m_{(3i+5) \wedge 15} + T_{i+33}) \lhd S_3(i \wedge 3),$$

Where $S_3(0) = 4$, $S_3(1) = 11$, $S_3(2) = 16$, $S_3(3) = 23$, so ↵s cycles over the values 4, 11, 16, 23. This is a different S3 from MD4. We can write out the first few steps of the pass as follows:

$d_0 = d_1 + (d_0 + H(d_1, d_2, d_3) + m_1 + T_{33}) \lhd 4;$

$d_3 = d_0 + (d_3 + H(d_0, d_1, d_2) + m_6 + T_{34}) \lhd 11;$

$d_2 = d_3 + (d_1 + H(d_3, d_0, d_1) + m_{11} + T_{35}) \lhd 16;$

$d_1 = d_2 + (d_1 + H(d_2, d_3, d_1) + m_0 + T_{36}) \lhd 23;$

$d_0 = d_1 + (d_0 + H(d_1, d_2, d_3) + m_5 + T_{37}) \lhd 4;$ and so on.

- **MD5 Message Digest Pass 4:**

This pass uses the function $I(x, y, z) = y \oplus (x \vee \sim z)$. A separate step is done for each of the 16 words of the message. For each integer i from 0 through 15,

$$d_{(-i) \wedge 3} = d_{(1-i) \wedge 3} + (d_{(-i) \wedge 3} + I(d_{(1-i) \wedge 3}, d_{(2-i) \wedge 3}, d_{(3-i) \wedge 3}) + m_{(7i) \wedge 15} + T_{i+49}) \lhd S_4(i \wedge 3),$$

Where $S_4(i) = (i+3)(i+4)/2$, so ↵s cycles over the values 6, 10, 15, 21. We can write out the first few steps of the pass as follows:

$d_0 = d_1 + (d_0 + I(d_1, d_2, d_3) + m_0 + T_{49}) \lhd 6;$

$d_3 = d_0 + (d_3 + I(d_0, d_1, d_2) + m_7 + T_{50}) \lhd 10;$

$d_2 = d_3 + (d_1 + I(d_3, d_0, d_1) + m_{14} + T_{51}) \lhd 15;$

$d_1 = d_2 + (d_1 + I(d_2, d_3, d_1) + m_5 + T_{52}) \lhd 21;$

$d_0 = d_1 + (d_0 + I(d_1, d_2, d_3) + m_{12} + T_{53}) \lhd 6;$ and so on.

MD4 AND MD5 HASHES

The 128-bit (16-byte) MD4, and MD5 hashes (also termed message digests) are typically represented as 32-digit hexadecimal numbers. The following demonstrates a 43-byte ASCII input and the corresponding MD2 hash:

MD4("The quick brown fox jumps over the lazy dog")

= 1bee69a46ba811185c194762abaeae90

MD5("The quick brown fox jumps over the lazy dog")
= 9e107d9d372bb6826bd81d3542a419d6

Even a small change in the message will (with overwhelming probability) result in a completely different hash, due to the avalanche effect. For example, changing d to c:

MD4("The quick brown fox jumps over the lazy cog")
= b86e130ce7028da59e672d56ad0113df

MD5("The quick brown fox jumps over the lazy cog")
= 1055d3e698d289f2af8663725127bd4b

The hash of the zero-length string is:

MD4("") = 31d6cfe0d16ae931b73c59d7e0c089c0

MD5("") = d41d8cd98f00b204e9800998ecf8427e

SECURE HASH STANDARD (SHS)

The Secure Hash Standard (SHS) is a set of cryptographically secure hash algorithms specified by the National Institute of Standards and Technology. This standard specifies a number of Secure Hash Algorithms (SHA), SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256. All of the algorithms are iterative, one-way hash functions that can process a message to produce a condensed representation called a **message digest**. These algorithms enable the determination of a message's integrity: any change to the message will, with a very high probability, results in a different message digest. This property is useful in the generation and verification of digital signatures and message authentication codes, and in the generation of random numbers or bits.

Each algorithm can be described in two stages: **preprocessing** and **hash computation**. Preprocessing involves padding a message, parsing the padded message into m-bit blocks, and setting initialization values to be used in the hash computation. The hash computation generates a **message schedule** from the padded message and uses that schedule, along with functions, constants, and word operations to iteratively generate a series of hash values. The final hash value generated by the hash computation is used to determine the message digest.

The algorithms differ most significantly in the security strengths that are provided for the data being hashed. The security strengths of these hash functions and the system as a whole when each of them is used with other cryptographic algorithms, such as digital signature algorithms and keyed-hash message authentication codes, can be found in [SP 800-57] and [SP 800-107].

Additionally, the algorithms differ in terms of the size of the blocks and words of data that are used during hashing or message digest sizes. Figure 4.1 presents the basic properties of these hash algorithms.

Table 4.1: Secure Hash Algorithm Properties

	SHA-1	SHA -2			
	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message Size (bits)	$<2^{64}$	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
Block Size (bits)	512	512	512	1024	1024
Word Size (bits)	32	32	32	64	64
Message Digest Size (bits)	160	224	256	384	512
Number of Steps (Iteration per block)	80	64	64	80	80

SHA-1 (Secure Hash Algorithm 1)

SHA-1 (**secure hash algorithm**) was proposed by NIST as messages digest function. SHA-1 takes a message of length at most 264 bits and produces a 160-bit output. It is similar to the MD5 message digest function, but it is a little slower to execute and presumably more secure. MD4 made three passes over each block of data; MD5 made four; SHA-1 makes five. It also produces a 160-bit digest as opposed to the 128 of the MDs.

OPERATIONS

- **SHA -1 Message Padding**

SHA-1 pads messages in the same manner as MD4 and MD5, except that SHA-1 is not defined for a message that is longer than 264 bits.

- **Overview SHA-1 Message Digest Computation**

Just like MD4 and MD5, SHA-1 operates in stages (see figure 4.5). Each stage mangles the pre-stage message digest by a sequence of operations based on the current message block. At the end of the stage, each word of the mangled message digest is added to its pre-stage value to produce the post-stage value (which becomes the pre-stage value for the next stage). Therefore, the current value of the message digest must be saved at the beginning of the stage so that it can be added in at the end of the stage.

The 160-bit message digest consists of five 32-bit words. Let's call them A, B, C, D, and E. Before the first stage they are set to A = 67452301_{16}, B = $efcdab89_{16}$, C = $98badcfe_{16}$, D = 10325476_{16}, E = $c3d2e1f0_{16}$. After the last stage, the value of A|B|C|D|E is the message digest for the entire message.

- **SHA-1 operations on a 512-bits block**

At the start of each stage, the 512-bit message block is used to create a 5×512-bit chunk (see figure 4.7). The first 512 bits of the chunk consist of the message block. The rest gets filled in, a 32-bit word at a time, according to the bizarre rule that the nth word (starting from word 16, since words 0 through 15 consist of the 512-bit message block) is the ⊕ of words n-3, n-8, n-14, and n-16. In SHA-1, the ⊕ of words n-3, n-8, n-14, and n-16 is rotated left one bit before being stored as word n; this is the only modification from the original SHA.

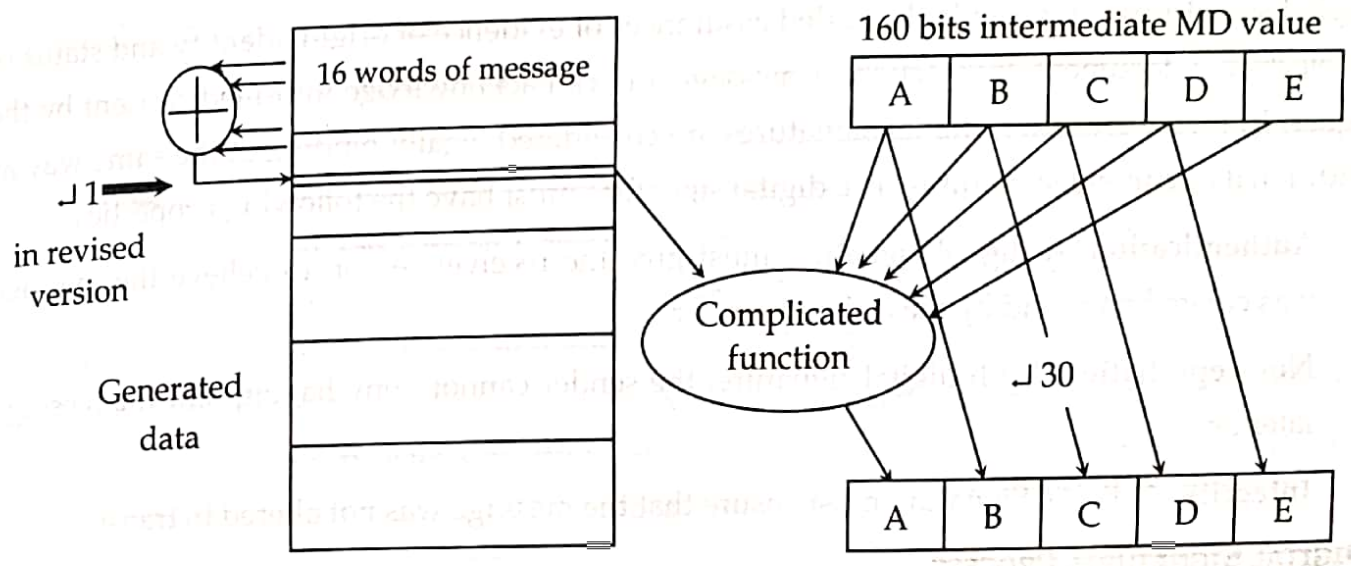


Figure 4.7: Inner Loop of SHA-1 (80 Iterations per Block)

Now we have a buffer of eighty 32-bit words (5x512 bits). Let's call the eighty 32-bit words W_0, W_1, ... , W_{79}. Now, a little program:

For t = 0 through 79, modify A, B, C, D, and E as follows:

B = old A C = old B ⌟30 D = old C E = old D

$A = E + (A \lrcorner 5) + W_t + K_t + f(t,B,C,D)$

In the above computation, everything to the right of the equal refers to the old values of A, B, C, D, and E. Since the new A depends on the old A, B, C, D, and E, a programmer would first compute the new A into a temporary variable V, and then after computing the new values of B, C, D, and E, set the new A equal to V. In the calculation of A, W_t is the t^{th} 32-bits word block and K_t is the constant depending upon the value of t given by the following relations:

$K_t = \lfloor 2^{30}\sqrt{2} \rfloor = 5a827999_{16}$ $(0 \le t \le 19)$

$K_t = \lfloor 2^{30}\sqrt{3} \rfloor = 6ed9eba1_{16}$ $(20 \le t \le 39)$

$K_t = \lfloor 2^{30}\sqrt{5} \rfloor = 8f1bbcdc_{16}$ $(40 \le t \le 59)$

$K_t = \lfloor 2^{30}\sqrt{10} \rfloor = ca62c1d6_{16}$ $(60 \le t \le 79)$

Again, f(t,B,C,D) is a function that varies according to the following relations:

$f(t,B,C,D) = (B \wedge C) \vee (\sim B \wedge D)$ $(0 \le t \le 19)$

$f(t,B,C,D) = B \oplus C \oplus D$ $(20 \le t \le 39)$

$f(t,B,C,D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$ $(40 \le t \le 59)$

$f(t,B,C,D) = B \oplus C \oplus D$ $(60 \le t \le 79)$

DIGITAL SIGNATURES

A **digital signature** is a mathematical technique used to validate the authenticity and integrity of a message, software or digital document. As the digital equivalent of a handwritten signature or stamped seal, a digital signature offers far more inherent security, and it is intended to solve the problem of tampering and impersonation in digital communications.

Digital signatures can provide the added assurances of evidence of origin, identity and status of an electronic document, transaction or message and can acknowledge informed consent by the signer. In many countries, digital signatures are considered legally binding in the same way as traditional document signatures. The digital signature must have the following properties:

- **Authentication:** A digital signature must give the receiver reason to believe the message was created and send by the claimed sender.
- **Non-repudiation:** With digital signature, the sender cannot deny having sent the message later on.
- **Integrity:** A digital Signature must ensure that the message was not altered in transit.

DIGITAL SIGNATURES PROCESS

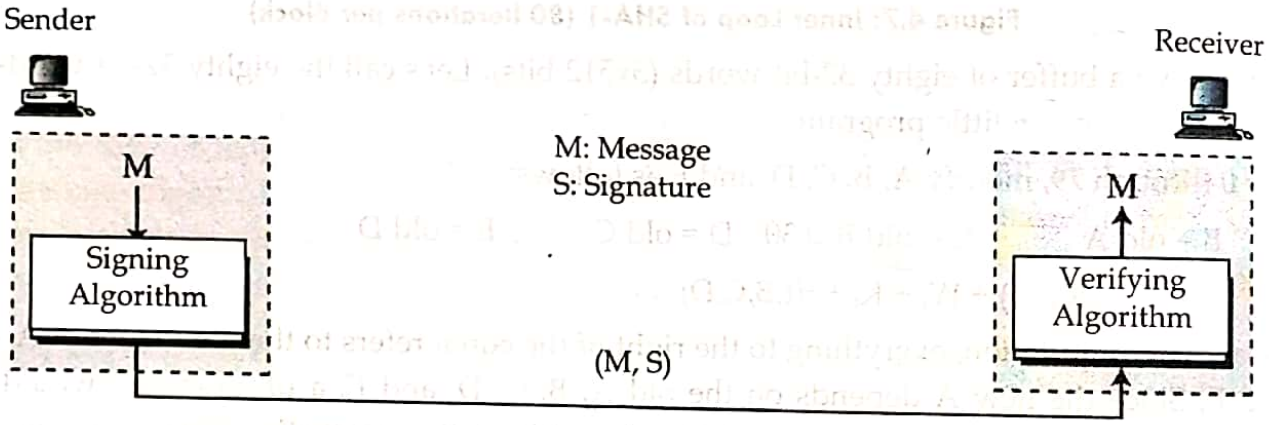


Figure 4.8: Digital Signature Process

Figure 4.8 shows the digital signature process. The sender uses a **signing algorithm** to sign the message. The sender uses a signing algorithm to sign the message. The message and the signature are sent to the receiver. The receiver receives the message and the signature and applies the **verifying algorithm** to the combination. If the result is true, the message is accepted; otherwise, it is rejected.

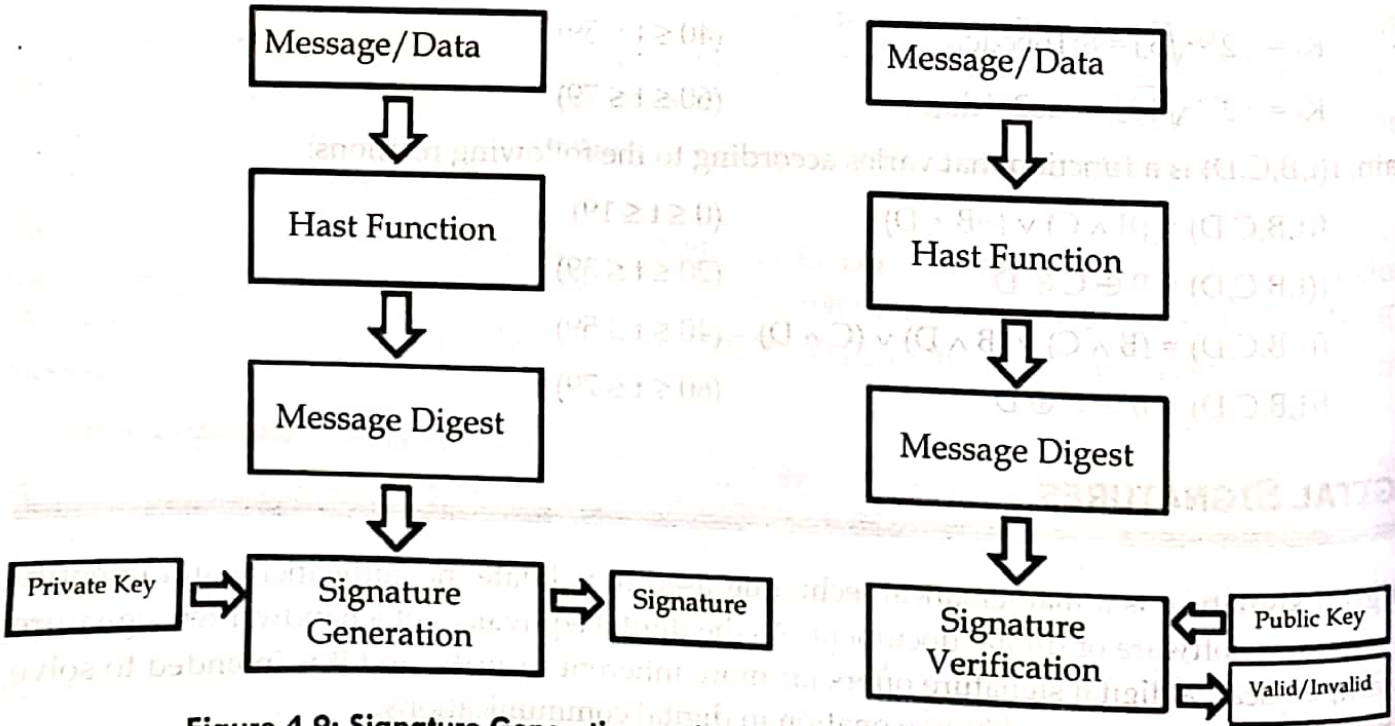


Figure 4.9: Signature Generation and Verification in Digital Signature Process

DIRECT DIGITAL SIGNATURES

Direct Digital Signatures involve the direct application of public-key algorithms involving only the communicating parties. A digital signature may be formed by encrypting the entire message with the sender's private key, or by encrypting a hash code of the message with the sender's private key. Confidentiality can be provided by further encrypting the entire message plus signature using either public or private key schemes. It is important to perform the signature function first and then an outer confidentiality function, since in case of dispute, some third party must view the message and its signature. But these approaches are dependent on the security of the sender's private-key. Will have problems if it is lost /stolen and signatures forged. Need time-stamps and timely key revocation.

ARBITRATED DIGITAL SIGNATURES

The problems associated with direct digital signatures can be addressed by using an arbiter, in a variety of possible arrangements. The arbiter plays a sensitive and crucial role in this sort of scheme, and all parties must have a great deal of trust that the arbitration mechanism is working properly. These schemes can be implemented with either private or public-key algorithms, and the arbiter may or may not see the actual message contents.

BENEFITS OF DIGITAL SIGNATURES

- **Authentication:** Digital signatures can be used to authenticate the source of messages. When ownership of a digital signature secret key is bound to a specific user, a valid signature shows that the message was sent by that user. For example, suppose a bank's branch office sends instructions to the central office requesting a change in the balance of an account. If the central office is not convinced that such a message is truly sent from an authorized source, acting on such a request could be a grave mistake.
- **Integrity:** In many cases, the sender and receiver of a message may have a need for trust that the message has not been altered during transmission. Although encryption hides the contents of a message, it may be possible to change an encrypted message without understanding it. However, if a message is digitally signed, any change in the message will invalidate the signature. Furthermore, there is no efficient way to modify a message and its signature to produce a new message with a valid signature, because this is still considered to be computationally infeasible by most cryptographic hash functions.

DRAWBACKS OF DIGITAL SIGNATURES

- **Trusted Time Stamping:** Digital signature algorithms and protocols do not inherently provide certainty about the date and time at which the underlying document was signed. The signer might, or might not, have included a time stamp with the signature, or the document itself might have a date mentioned on it, but a later reader cannot be certain the signer did not, for instance, backdate the date or time of the signature.

- **Non-repudiation:** The word repudiation refers to any act of disclaiming responsibility for a message. A message's recipient may insist the sender attach a signature in order to make later repudiation more difficult, since the recipient can show the signed message to a third party (e.g., a court) to reinforce a claim as to its signatories and integrity. However, loss of control over a user's private key will mean that all digital signatures using that key, and so ostensibly 'from' that user, are suspect. Nonetheless, a user cannot repudiate a signed message without repudiating their signature key. It is aggravated by the fact there is no trusted time stamp, so new documents (after the key compromise) cannot be separated from old ones, further complicating signature key invalidation. Certificate Authorities usually maintain a public repository of public-key so the association user-key is certified and signatures cannot be repudiated. Expired certificates are normally removed from the directory. It is a matter for the security policy and the responsibility of the authority to keep old certificates for a period of time if a non-repudiation of data service is provided.

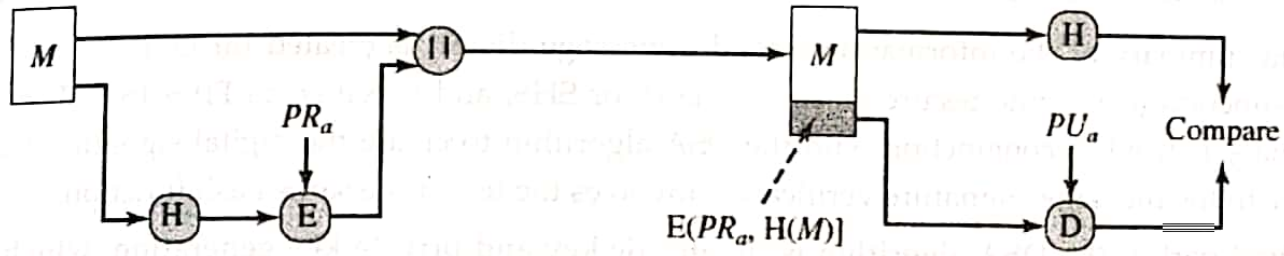
DIGITAL SIGNATURE STANDARD (DSS)

The National Institute of Standards and Technology (NIST) have published Federal Information Processing Standard FIPS 186, known as the Digital Signature Standard (DSS). The **Digital Signature Standard** is intended to be used in electronic funds transfer, software distribution, electronic mail, data storage and applications which require high data integrity assurance. The Digital Signature Standard can be implemented in software, hardware or firmware. The DSS makes use of the Secure Hash Algorithm (SHA), and presents a new digital signature technique, the **Digital Signature Algorithm (DSA)**. The DSS was originally proposed in 1991 and revised in 1993 in response to public feedback concerning the security of the scheme. There was a further minor revision in 1996. In 2000, an expanded version of the standard was issued as FIPS 186-2, which incorporates digital signature algorithms based on RSA and on elliptic curve cryptography.

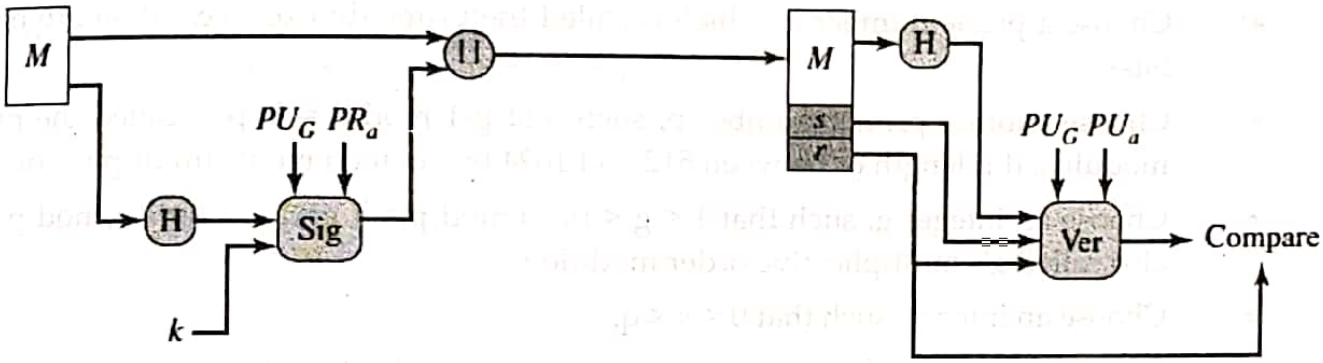
THE DSS APPROACH

The DSS uses an algorithm that is designed to provide only the digital signature function. Unlike RSA, it cannot be used for encryption or key exchange. Nevertheless, it is a public-key technique. Figure 4.10 contrasts the DSS approach for generating digital signatures to that used with RSA. In **the RSA approach**, the message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender's private key to form the signature. Both the message and the signature are then transmitted. The recipient takes the message and produces a hash code. The recipient also decrypts the signature using the sender's public key. If the calculated hash code matches the decrypted signature, the

signature is accepted as valid. Because only the sender knows the private key, only the sender could have produced a valid signature.



(a) RSA approach



(b) DSS approach

Figure 4.10: Two Approaches to Digital Signatures

The DSS approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number generated for this particular signature. The signature function also depends on the sender's private key and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key. The result is a signature consisting of two components, labeled *s* and *r*.

At the receiving end, the hash code of the incoming message is generated. This plus the signature is input to a verification function. The verification function also depends on the global public key as well as the sender's public key, which is paired with the sender's private key. The output of the verification function is a value that is equal to the signature component if the signature is valid. The signature function is such that only the sender, with knowledge of the private key, could have produced the valid signature.

THE DIGITAL SIGNATURE ALGORITHM

DSA is one of the many algorithms that are used to create digital signatures for data transmission. DSA is a pair of large numbers that are computed according to the specified algorithm within parameters that enable the authentication of the signatory, and as a consequence, the integrity of the data attached. Digital signatures are generated through DSA, as well as verified. Signatures are generated in conjunction with the use of a private key; verification takes place in reference to a corresponding public key. Each signatory has their own

paired public (assumed to be known to the general public) and private (known only to the user) keys. Because a signature can only be generated by an authorized person using their private key, the corresponding public key can be used by anyone to verify the signature.

A data summary of the information (called a message digest) is created through the use of a hash function (called the Secure Hash Standard, or SHS, and specified in FIPS 180). The data summary is used in conjunction with the DSA algorithm to create the digital signature that is sent with the message. Signature verification involves the use of the same hash function.

The first part of the DSA algorithm is the public key and private key generation, which can be described as:

- Choose a prime number q, which is called the prime divisor. (i.e. bit length of 160 bits)
- Choose another primer number p, such that p-1 mod q = 0. p is called the prime modulus. (bit length of between 512 and 1024 bits in increments (multiples) of 64)
- Choose an integer g, such that $1 < g < p$, $g^q \bmod p = 1$ and $g = h^{(p-1)/q} \bmod p$. q is also called g's multiplicative order modulo p.
- Choose an integer, such that $0 < x < q$.
- Compute $y = g^x \bmod p$.
- Package the public key as {p, q, g, y}.
- Package the private key as {p, q, g, x}.

The second part of the DSA algorithm is the signature generation and signature verification, which can be described as:

To generate a message signature, the sender can follow these steps:

- Generate the message digest h, using a hash algorithms like SHA-1.
- Generate a random number k, such that $0 < k < q$.
- Compute $r = (g^k \bmod p) \bmod q$. If r = 0, select a different k.
- Compute i, such that $k^i \bmod q = 1$. i is called the modular multiplicative inverse of k modulo q.
- Compute $s = i(h+rx) \bmod q$. If s = 0, select a different k.
- Package the digital signature as {r, s}.

To verify a message signature, the receiver of the message and the digital signature can follow these steps:

- Generate the message digest h, using the same hash algorithm.
- Compute w, such that $sw \bmod q = 1$. w is called the modular multiplicative inverse of s modulo q.
- Compute $u1 = h \times w \bmod q$.
- Compute $u2 = r \times w \bmod q$.
- Compute $v = ((g^{u1} \times y^{u2}) \bmod p) \bmod q$.
- If v == r, the digital signature is valid.

EXAMPLE: To demonstrate the DSA digital signature algorithm, let's try it with a smaller prime divisor q=3 and prime modulus p=7.

The process of generating the public key and private key can be illustrated as:

q = 3
p = 7
g = 4
x = 5
y = 2
{p, q, g, y} = {7, 3, 4, 2} (i.e. public key)
{p, q, g, x} = {7, 3, 4, 5} (i.e. private key)

With the private key {p, q, g, x} = {7, 3, 4, 5}, the process of generating a digital signature out a message hash value of h=3 can be illustrated as:

h = 3
k = 2
r = 2
i = 5
s = 2
{2, 2} (i.e., digital signature)

The process of verifying the digital signature {r, s}={2, 2} with the public key {p, q, g, y} = {7, 3, 4, 2} can be illustrated as:

h = 3
w = 5
u1 = 0
u2 = 1
v = 2
v == r (i.e. verification passed)

Advantages of Digital Signature Algorithm

- Along with having strong strength levels, the length of the signature is smaller as compared to other digital signature standards.
- The signature computation speed is less.
- DSA requires less storage to work as compared to other digital standards.
- DSA is patent free so it can be used free of cost.

Disadvantages of Digital Signature Algorithm

- It requires a lot of time to authenticate as the verification process includes complicated remainder operators. It requires a lot of time for computation.
- Data in DSA is not encrypted. We can only authenticate data in this.
- The digital signature algorithm firstly computes with SHA1 hash and signs it. Any drawbacks in cryptographic security of SHA1 are reflected in DSA because implicitly of DSA is dependent on it.
- With applications in both secret and non-secret communications, DSA is of the US National Standard.

Discussion Exercise

1. What do you mean by message authentication?
2. What types of attacks are addressed by message authentication?
3. Explain the message authentication functions briefly.
4. Discuss the message authentication code (MAC) in detail.
5. What do you mean by cryptographic hash function?
6. List the properties of cryptographic hash function. Explain each of them.
7. What are the applications of cryptographic hash functions? Explain.
8. Why hash function is used for the password verification rather than plain text
9. Define message digest with example.
10. Differentiate between MD4 and MD5.
11. Explain the SHA-1 in detail.
12. Differentiate between MD5 and SHA-1.
13. What characteristics are needed in secure hash function?
14. What are the properties a digital signature should have?
15. What is the difference between direct and arbitrated digital signature?
16. Explain the signature generation and signature verification in digital signature process.
17. List the pros and cons of digital signature.
18. Discuss the two approaches of DSS.
19. Explain the DSA algorithm.
20. Demonstrate DSA digital signature algorithm, with the help of prime divisor q=11 and prime modulus p = 23.

❑❑❑