CHAPTER 4: CRYPTOGRAPHIC HASH FUNCTIONS AND DIGITAL SIGNATURES

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

MESSAGE AUTHENTICATION(SECURITY) REQUIREMENTS

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

1. **Disclosure:** Release of message contents to any person or process not possessing the appropriate cryptographic key.
2. **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.
3. **Masquerade:** Insertion of messages into the network from a fraudulent source.

MESSAGE AUTHENTICATION(SECURITY) REQUIREMENTS

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.

4. **Content modification:** Changes to the contents of a message, including insertion, deletion, transposition, and modification.
5. **Sequence modification:** Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.
6. **Timing modification:** Delay or replay of messages.

MESSAGE AUTHENTICATION(SECURITY) REQUIREMENTS

7. **Source repudiation:** Denial of transmission of message by source.
8. **Destination repudiation:** Denial of receipt of message by destination.

HOW TO DEAL WITH THESE?

- Measures to deal with the first two attacks are in the realm of message confidentiality.
- Measures to deal with items (3) through (6) in the foregoing list are generally regarded as message authentication.
- Mechanisms for dealing specifically with item (7) come under the heading of digital signatures. Generally, a digital signature technique will also counter some or all of the attacks listed under items (3) through (6).
- Dealing with item (8) may require a combination of the use of digital signatures and a protocol designed to counter this attack.

CONCLUSION

- 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

Types of functions that may be used to produce an authenticator. These 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 ENCRYPTION

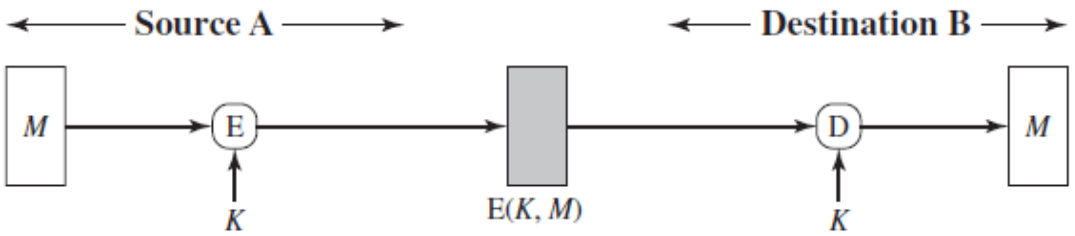
Message encryption by itself can provide a measure of authentication.

SYMMETRIC ENCRYPTION

PUBLIC-KEY ENCRYPTION

SYMMETRIC ENCRYPTION

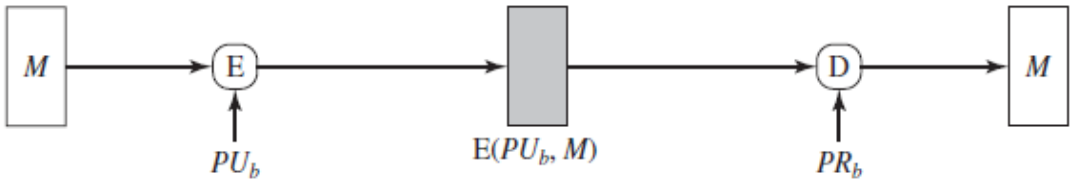
- A message transmitted from source A to destination B is encrypted using a secret key shared by A and B. If no other party knows the key, then confidentiality is provided: No other party can recover the plaintext of the message.
- In addition, B is assured that the message was generated by A. Why? The message must have come from A, because A is the only other party that possesses K and therefore the only other party with the information necessary to construct ciphertext that can be decrypted with K.



(a) Symmetric encryption: confidentiality and authentication

PUBLIC-KEY ENCRYPTION

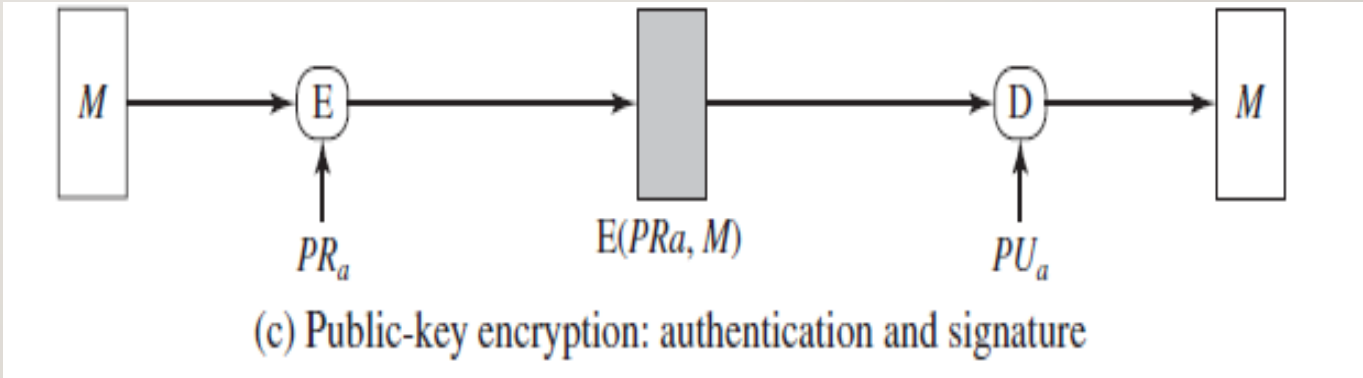
- The straightforward use of public-key encryption (fig.b) provides confidentiality but not authentication. The source (A) uses the public key PU_b of the destination (B) to encrypt M. Because only B has the corresponding private key PR_b , only B can decrypt the message. This scheme provides no authentication, because any opponent could also use B's public key to encrypt a message and claim to be A.



(b) Public-key encryption: confidentiality

HOW TO PROVIDE AUTHENTICATION WITH PUBLIC KEY?

- To provide authentication, A uses its private key to encrypt the message, and B uses A's public key to decrypt (Figure. c).This provides authentication using the same type of reasoning as in the symmetric encryption case: The message must have come from A because A is the only party that possesses PRa and therefore the only party with the information necessary to construct ciphertext that can be decrypted with PUa .



(c) Public-key encryption: authentication and signature

TO PROVIDE BOTH CONFIDENTIALITY AND AUTHENTICATION USING PUBLIC KEY

- To provide both confidentiality and authentication, A can encrypt M first using its private key, which provides the digital signature, and then using B's public key, which provides confidentiality (Figure d).
- The disadvantage of this approach is that the public-key algorithm, which is complex, must be exercised four times rather than two in each communication.

(d) Public-key encryption: confidentiality, authentication, and signature

MESSAGE AUTHENTICATION CODE

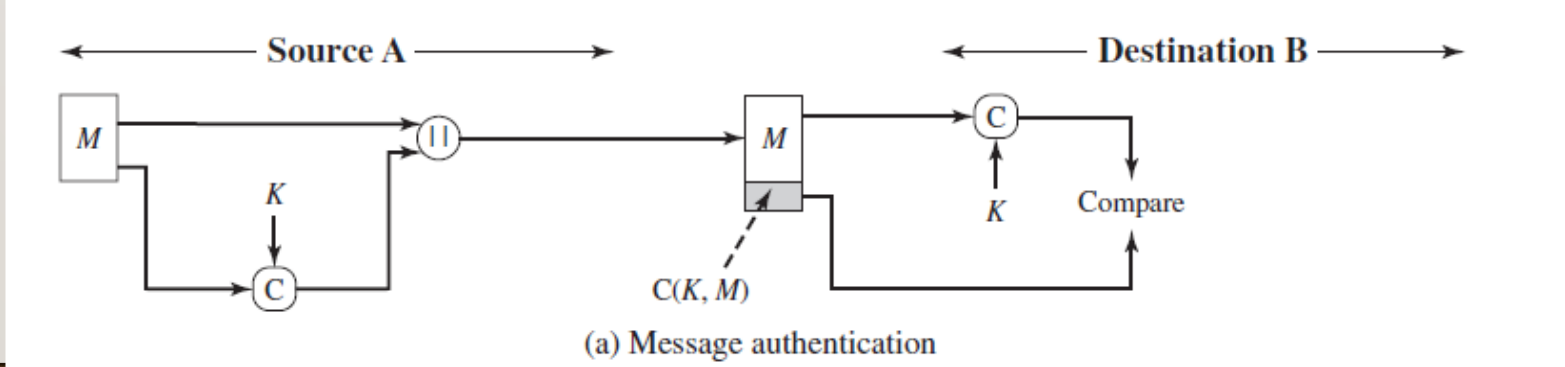
- An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as a **cryptographic checksum** or MAC, that is appended to the message.
- This technique assumes that two communicating parties, say A and B, share a common secret key K. When A has a message to send to B, it calculates the MAC as a function of the message and the key:

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

Where M= Input message , K=shared secret key, C=MAC function, MAC=Message Authentication Code

MESSAGE AUTHENTICATION CODE

- So, Generated by an algorithm that creates a small fixed-sized blocks.
- Depend on both message and some key.
- Like encryption though need not be reversible.
- The message plus MAC are transmitted to the intended recipient
- Receiver performs same computation on the received message, using the same secret key to generate a new MAC. The received MAC is compared to the calculated MAC (Figure a).



(a) Message authentication

If we assume that only the receiver and the sender know the identity of the secret key, and if the received MAC matches the calculated MAC, then

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

3. If the message includes a sequence number (such as is used with HDLC,X.25, and TCP), then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number.

MAC is many to one function i.e. many messages have same MAC.

DIGITAL SIGNATURES

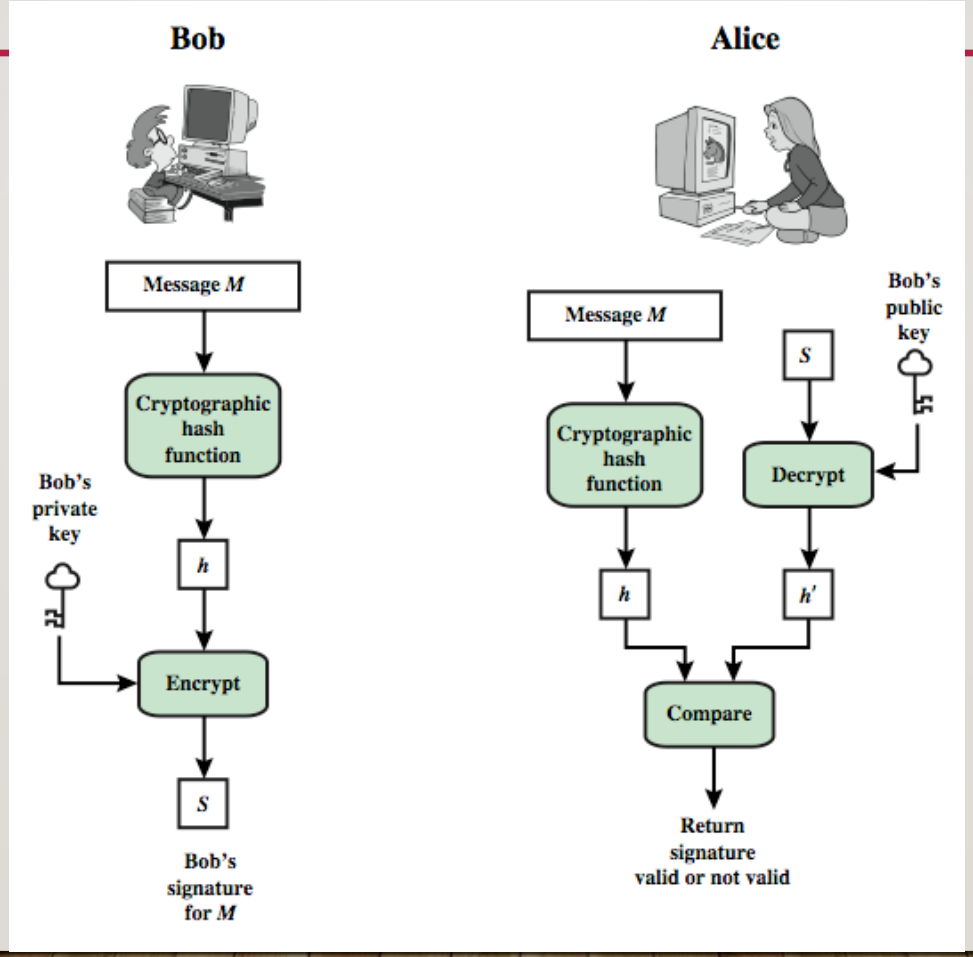
- The most important development from the work on public-key cryptography is the digital signature.
- Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other i.e. have looked at message authentication but does not address issues of lack of trust.
- Digital signatures provide the ability to:
  - verify author, date & time of signature.
  - authenticate message contents .
  - be verified by third parties to resolve disputes.

  Hence include authentication function with additional capabilities

Digital signatures employ a type of Asymmetric Cryptography. The Scheme typically consists of three Algorithms

- A key generation algorithm that selects a private key uniformly at random from a set of possible private keys. The algorithm outputs the private key and a corresponding public key.
- A signing algorithm that, given a message and a private key, produces a signature.
- A signature verifying algorithm that, given a message, public key and a signature, either accepts or rejects the message's claim to authenticity

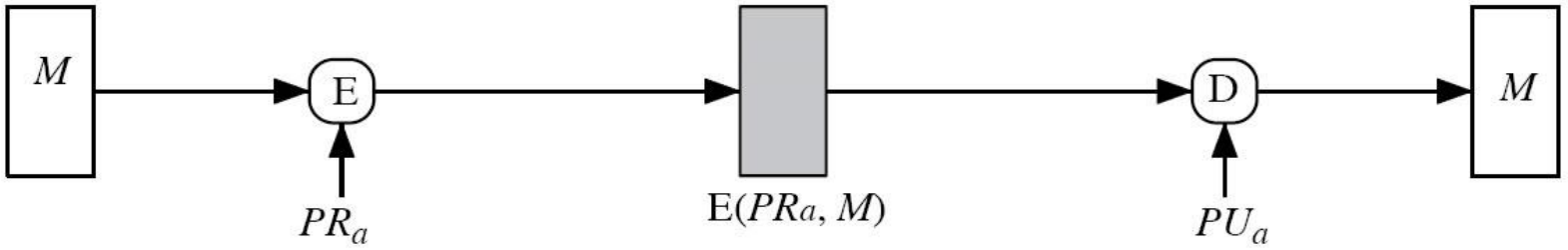
DIGITAL SIGNATURE MODEL DIGITAL SIGNATURE MODEL



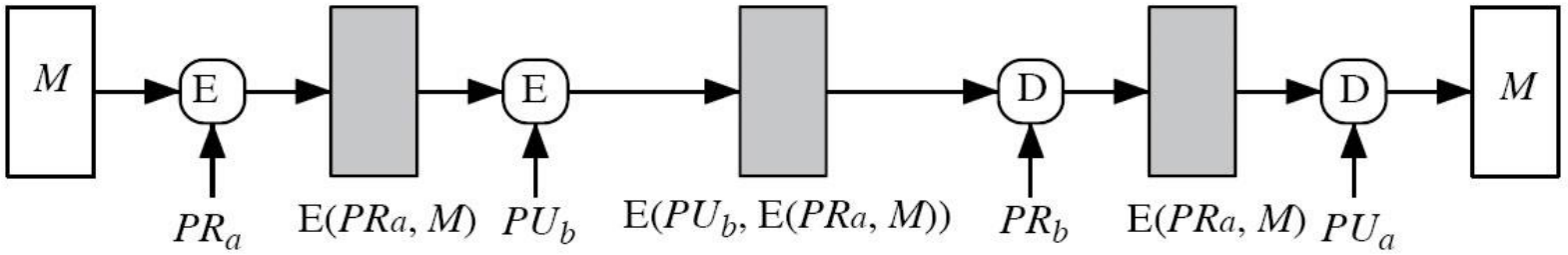
DIRECT DIGITAL SIGNATURES

- Direct Digital Signatures involve the direct application of public-key algorithms involving only the communicating parties. (involve only sender & receiver).
- 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.

DIRECT DIGITAL SIGNATURES



(c) Public-key encryption: authentication and signature



(d) Public-key encryption: confidentiality, authentication, and signature

Weakness: Security depends on sender's private-key

DIRECT DIGITAL SIGNATURES

- Problems with direct signatures:
  - ✓ Validity of scheme depends on the security of the sender's private key ⇒ sender may later deny sending a certain message.
  - ✓ Private key may actually be stolen from X at time T, so timestamp may not help.

ARBITRATED DIGITAL SIGNATURES

- In the arbitrated signature scheme, there is a trusted third party called the arbiter.
- Every signed message from a sender X to a receiver Y goes first to an arbiter A, who subjects the message and its signature to a number of tests to check its origin and content.
- The message is then dated and sent to Y with an indication that it has been verified to the satisfaction of the arbiter.
- The presence of A solves the problem faced by direct signature schemes, namely that X might deny sending a message. The arbiter plays a sensitive and crucial role in this scheme, and all parties must trust that the arbitration mechanism is working properly.

Arbitrated Digital Signatures

(1) $X \rightarrow A$: $ID_X \| E(PR_x, [ID_X \| E(PU_y, E(PR_x, M))])$
(2) $A \rightarrow Y$: $E(PR_a, [ID_X \| E(PU_y, E(PR_x, M)) \| T])$

(c) Public-Key Encryption, Arbiter Does Not See Message

Notations:

X=sender
Y=recipient
A=Arbiter
ID_X=ID of X

M=message
T=time stamp
PR_X=X's private key
PU_Y=Y's public key
PR_A=A's private key

Weakness: twice public-key encryptions on the message

DIGITAL SIGNATURE REQUIREMENTS

- Must depend on the message signed.
- Must use information unique to sender
- Must be relatively easy to produce
- Must be relatively easy to recognize & verify
  - Directed : Recipient can verify
  - Arbitrated: Anyone can verify
  - Be computationally infeasible to forge
  - Be able to retain a copy of the signature in storage

DIGITAL SIGNATURE STANDARD (DSS)

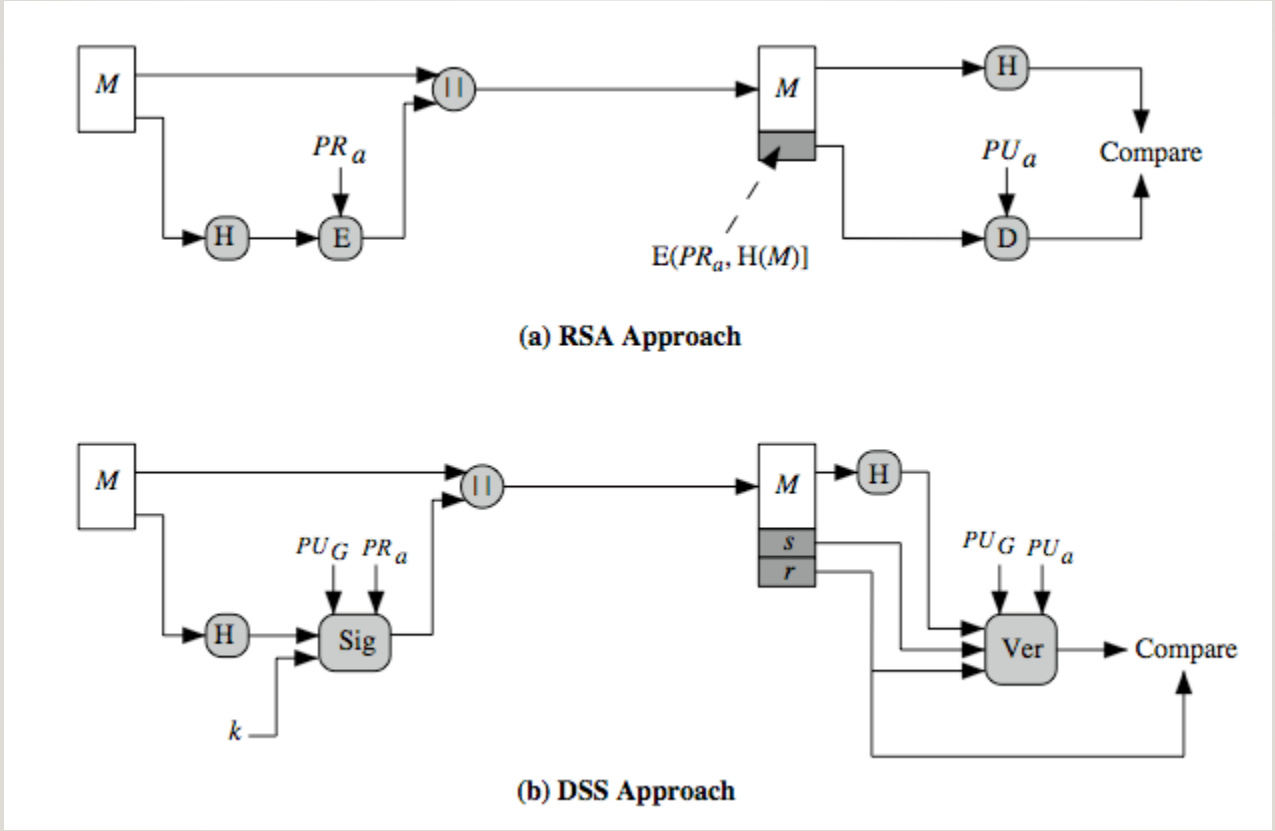
- US Govt approved signature scheme
- Designed by NIST & NSA in early 90's
- Published as FIPS-186 in 1991
- Revised in 1993, 1996 & then 2000
- Uses the SHA hash algorithm
- DSS is the standard, DSA is the algorithm

A PUBLIC VERSUS A PRIVATE APPROACH TO DIGITAL SIGNATURES

- Another way of classifying digital signature schemes is based on whether a private-key system or a public-key system is used.
- The public-key system based digital signatures have several advantages over the private-key system based digital signatures.
- **The two most popular and commonly used public-key system based digital signature schemes are the RSA (named after Rivest, Shamir, and Aldeman, the inventors of the RSA public-key encryption scheme) and the digital signature algorithm (DSA) approaches.**

- The DSA is incorporated into the Digital Signature Standard (DSS), which was published by the National Institute of Standards and Technology as the Federal Information Processing Standard. It was first proposed in 1991, revised in 1993, and further revised with minor changes in 1996.

DSS VS. RSA SIGNATURES



31 RSA SCHEME

- 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.

DSS

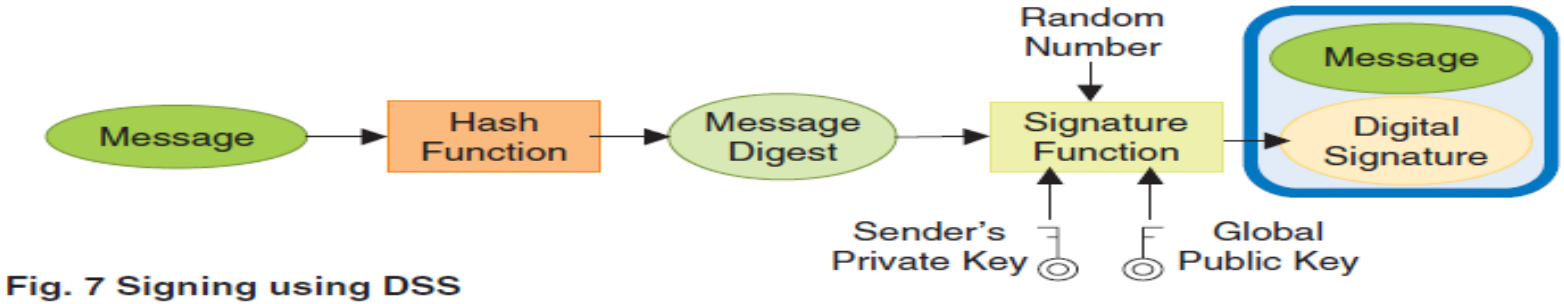


Fig. 7 Signing using DSS

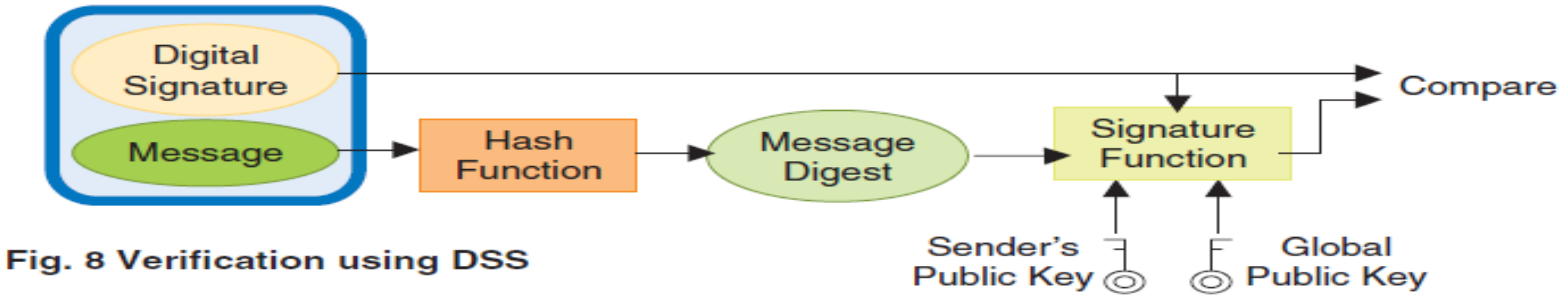


Fig. 8 Verification using DSS

DSS SCHEME

- makes use of a hash function.
- The hash code is provided as input to a signature function along with a random number k generated for this particular signature.
- The signature function also depends on the sender's private key (PRa)and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key (PU_G).

34 DSS Scheme

- 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 (PUa), 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 r 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)

Global Public-Key Components	
p	A prime number of L bits where L is a multiple of 64 and $512 \le L \le 1024$
q	A 160-bit prime factor of p-1
g	$= h^{(p-1)/q} \bmod p$, where h is any integer with $1<h< p-1$, such that $(h^{(p-1)/q} \bmod p)>1$
User's Private Key	
x	A random or pseudorandom integer with $0<x<q$
User's Public Key	
y	$= g^x \bmod p$
User's Per-Message Secret Number	
k	A random or pseudorandom integer with $0<k<q$
Signing	
$r = (g^k \bmod p) \bmod q$ $\quad$ $s = [k^{-1} (H(M) = xr)] \bmod q$ Signature = (r, s)	
Verifying	
$w = (s')^{-1} \bmod q$ $u_1 = [H(M')w] \bmod q$ $\quad$ $u_2 = (r')w \bmod q$ $\quad$ $v = [(g^{u1}y^{u2}) \bmod p] \bmod q$ Test: $v = r'$	

DSS

- DSA
  - M = message to be signed
  - H(M) = hash of M using SHA
  - M', r', s' = received versions of M, r, s

Example DSA

p = 23 # computed prime modulus: (p-1) mod q = 0
q=11, which is prime divisor/factor of p-1 i.e. (23-1=22)
g=4

DIGITAL SIGNATURES APPLICATIONS

- Digital signatures are being used in secure e-mail and credit card transactions over the Internet. The two most common secure e-mail systems using digital signatures are Pretty Good Privacy and Secure/Multipurpose Internet Mail Extension. Both of these systems support the RSA as well as the DSS-based signatures.
- The most widely used system for the credit card transactions over the Internet is Secure Electronic Transaction (SET). It consists of a set of security protocols and formats to enable prior existing credit card payment infrastructure to work on the Internet.

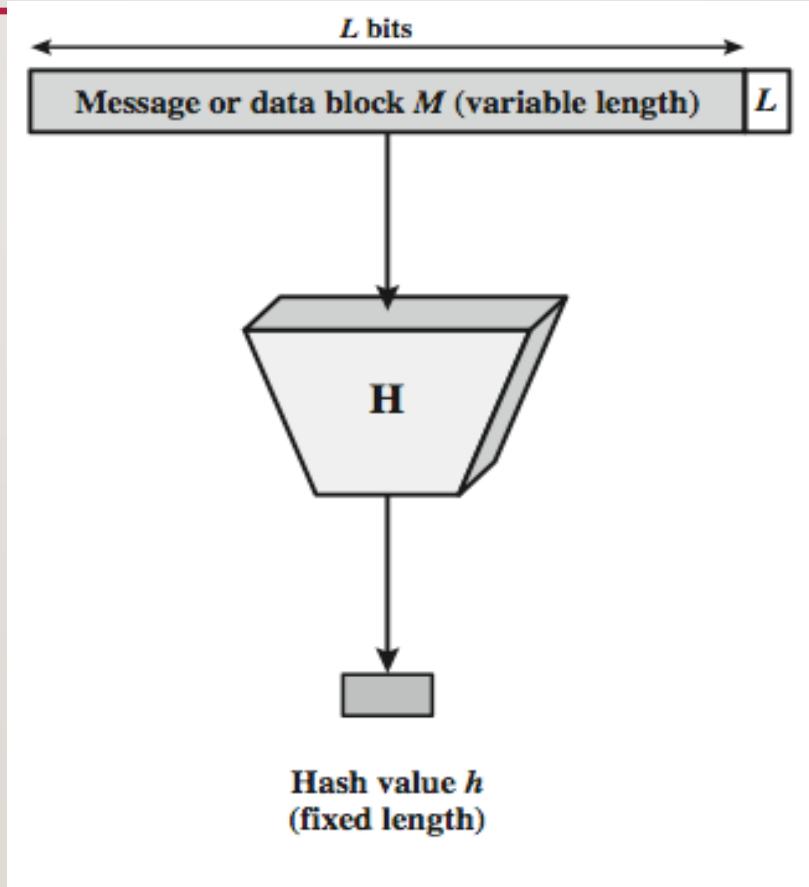
HASHES

- A hash value h is generated by a function H of the form h = H(M)

where M is a variable-length message and H(M) is the fixed-length hash value.

The hash value is appended to the message at the source at a time when the message is assumed or known to be correct.

The receiver authenticates that message by recomputing the hash value. Because the hash function itself is not considered to be secret, some means is required to protect the hash value


L bits
Message or data block M (variable length)
L
H
Hash value h
(fixed length)

REQUIREMENTS FOR A HASH FUNCTION

1. H can be applied to a block of data of any size.
2. H produces a fixed-length output.
3. H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical.
4. For any given value h, it is computationally infeasible to find x such that H(x) = h. This is sometimes referred to in the literature as **the one-way property.**
5. For any given block x, it is computationally infeasible to find $y \neq x$ such that H(y) = H(x). This is sometimes referred to as **weak collision resistance.**
6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y). This is sometimes referred to as **strong collision resistance**.

APPLICATIONS

- **Message Integrity Verification:** Determining whether any changes have been made to a message (or a file), for example, can be accomplished by comparing message digests calculated before, and after, transmission (or any other event).

Applications

- **Password Verification**: Passwords are usually not stored in cleartext, for obvious reasons, but instead in digest form. To authenticate a user, the password presented by the user is hashed and compared with the stored hash. This is sometimes referred to as one-way encryption.

- **Digital Signatures:**

  while generating digital signatures, the message digest is created and it is encrypted with the private key so that the signing process becomes faster.

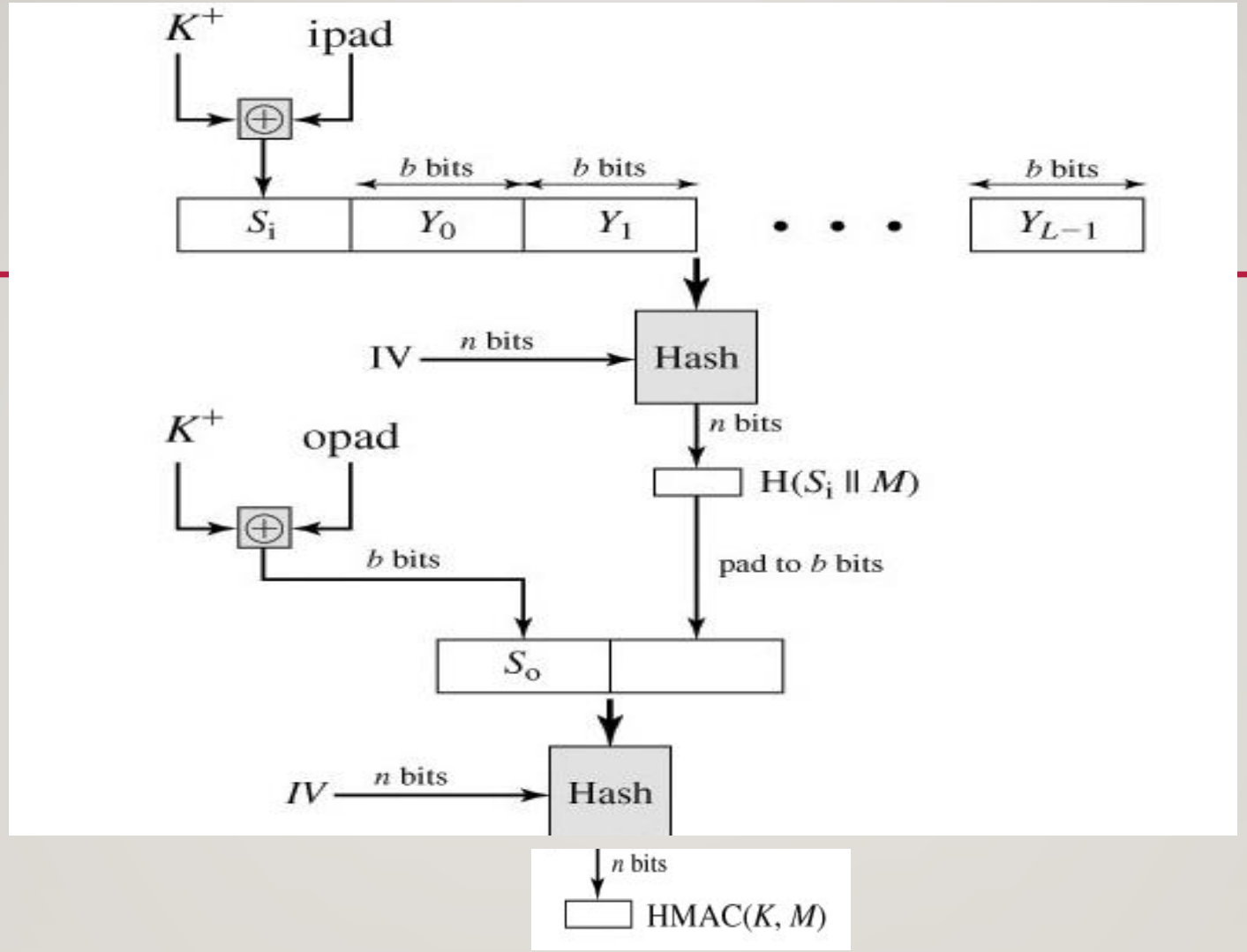
HMAC (KEYED-HASH MESSAGE AUTHENTICATION CODE)

- A keyed-Hash Message Authentication Code (HMAC or KHMAC), is a type of message authentication code (MAC) calculated using a specific algorithm involving a cryptographic **hash function in combination with a secret key**.

- It may be used to verify both the **data integrity and the authenticity of a message**.
- Any iterative cryptographic hash function, like MD5 or SHA-1, may be used in HMAC; the resulting MAC algorithm is termed HMAC-MD5 or HMAC-SHA-1 accordingly.
- The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, on the size and quality of the key and the size of the hash output length in bits.

HMAC

- A MAC derived from hash function is called HMAC.
- The reason for developing HMAC were that hash functions incur less overhead than encryption and the code of hash functions is easily and freely available.
- HMAC is a great resistant towards cryptanalysis attacks as it uses the Hashing concept twice.
- HMAC consists of twin benefits of Hashing and MAC, and thus is more secure than any other authentication codes.


K^+
ipad
b bits
b bits
b bits
S_i
Y_0
Y_1
Y_{L-1}
IV
n bits
Hash
n bits
K^+
opad
$H(S_i \| M)$
b bits
pad to b bits
S_o
IV
n bits
Hash
n bits
HMAC(K, M)

- H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)
- *IV* = initial value input to hash function.
- *M* = message input to HMAC (including the padding specified in the embedded hash function).
- Y_i = i^{th} block of *M*, $0 \leq i \leq (L - 1)$.
- *L* = number of blocks in *M*.
- *b* = number of bits in a block.
- *n* =length of hash code produced by embedded hash function.
- *K* = secret key; recommended length is *n*; if key length is greater than *b*,the key is input to the hash function to produce an *n*-bit key.

- *K*+ =*K* padded with zeros on the left so that the result is *b* bits in length.
- ipad =00110110 (36 in hexadecimal) repeated *b*/8 times.
- opad 01011100 (5C in hexadecimal) repeated *b*/8 times.

HOW ALGORITHM WORKS (DESCRIPTION)?

1. Append zeros to the left end of K to create a b-bit string K^+ .
2. XOR (bitwise exclusive-OR) K^+ with ipad to produce the b-bit block S_i .
3. Append M to S_i .
4. Apply H to the stream generated in step 3.
5. XOR K^+ with opad to produce the b-bit block S_0 .
6. Append the hash result from step 4 to S_0 .
7. Apply H to the stream generated in step 6 and output the result.

- Then HMAC can be expressed as follows:

$$HMAC(K,M) = H[(K^{+} \text{ XOR opad}) || H[(K^{+} \text{ XOR ipad}) || M]]$$

Examples of Hash Functions

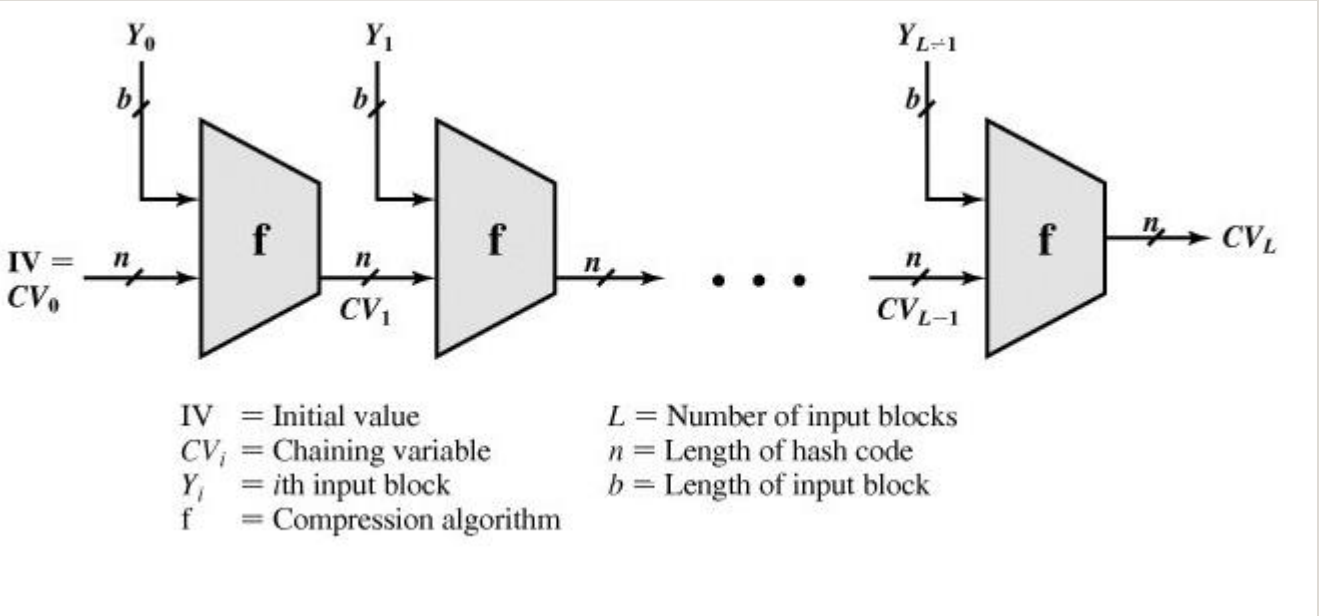
1. MD2
2. Snefru = Fast hash named after Egyptian king !
3. MD4 = Message Digest 4
4. Snefru 2 = Designed after Snefru was broken !
5. MD5 = Message Digest 5
6. SHA = Secure hash algorithm !
7. SHA-1 = Updated SHA !
8. SHA-2 = SHA-224, SHA-256, SHA-384, SHA-512 SHA-512 uses 64-bit operations !
9. HMAC = Keyed-Hash Message Authentication Code

DIFFERENT MESSAGE DIGEST ALGORITHMS

Table 12.1 *Characteristics of Secure Hash Algorithms (SHAs)*

Characteristics	*SHA-1*	*SHA-224*	*SHA-256*	*SHA-384*	*SHA-512*
Maximum Message size	$2^{64} - 1$	$2^{64} - 1$	$2^{64} - 1$	$2^{128} - 1$	$2^{128} - 1$
Block size	512	512	512	1024	1024
Message digest size	160	224	256	384	512
Number of rounds	80	64	64	80	80
Word size	32	32	32	64	64

GENERAL STRUCTURE OF SECURE HASH CODE



GENERAL STRUCTURE OF SECURE HASH CODE

- This structure, referred to as an iterated hash function, was proposed by Merkle and is the structure of most hash functions in use today, including SHA and Whirlpool.
- The hash function takes an input message and partitions it into L fixed-sized blocks of b bits each.
- If necessary, the final block is padded to b bits.
- The final block also includes the value of the total length of the input to the hash function.
- The inclusion of the length makes the job of the opponent more difficult. Either the opponent must find two messages of equal length that hash to the same value or two messages of differing lengths that, together with their length values, hash to the same value

- If the compression function is collision resistant, then the resultant iterated hash function is also collision resistant and hence secure.
- Thus, the problem of designing a secure hash function is that of designing a collision-resistant compression function operates on inputs of some fixed size.

SECURE HASH ALGORITHM

- **The Secure Hash Algorithm (SHA)** was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993; a revised version was issued as FIPS 180-1 in 1995 and is generally referred to as SHA-1.
- SHA-1 produces a hash value of 160 bits.
- In 2002, NIST produced a revised version of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512.
- These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1.

COMPARISON OF SHA PARAMETERS

	SHA-1	SHA-256	SHA-384	SHA-512
Message digest size	160	256	384	512
Message size	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
Block size	512	512	1024	1024
Word size	32	32	64	64
Number of steps	80	64	80	80

Note: All sizes are measured in bits.

MESSAGE DIGEST ALGORITHMS
MD 4 (MESSAGE DIGEST 4)

- MD 4 Algorithm is a cryptographic hash function developed by Ronald Rivest in 1990.
- The MD4 message digest algorithm takes an input message of arbitrary length and produces an output 128-bit (four 32 bits words) "fingerprint" or "message digest“.
- The MD4 algorithm is thus ideal for digital signature applications: a large file can be securely "compressed" with MD4 before being signed with (say) the RSA public-key cryptosystem.
- It is optimized for 32-bit computers.

TERMINOLOGY AND NOTATION

- A ***word*** is a 32-bit quantity and a byte is an 8-bit quantity.
- Let the symbol "+" denote addition of words (i.e., modulo- 2^{32} addition).
- Let ***(X*** <<< s) denote the 32-bit value obtained by circularly shifting (rotating) X left by s bit positions.
- Let ¬***X*** denote the bit-wise complement of X, and let X V *Y* denote the bit-wise OR of X and *Y* . Let X $\oplus$ *Y* denote the bit-wise **XOR** of ***X*** and ***Y*** **,** and let ***XY*** denote the bit-wise **AND** of *X* and *Y* .

STEP 1: 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 1 followed by required number of 0s so that the length of the message is 64 bits less than multiple of 512.
- The remaining 64 bits is used for providing length of the original message i.e. unpadded message.

Original Message	1000……..000	Original length in bits 64 bits

- Padding is always performed, even if the length of the message is already congruent to 448, modulo 512 (in which case 512 bits of padding are added).
- single "1" bit is appended to the message, and then enough zero bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512.

STEP 2. APPEND LENGTH

- A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step.
- These bits are appended as two 32-bit words and appended low-order word first in accordance with the previous conventions.
- At this point the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32-bit) words. Let *M[0…..**N -1**]* denote the words of the resulting message, where ***N*** is a multiple of 16.

STEP 3. INITIALIZE MD BUFFER

- A 4-word buffer (A, B , C, D) is used to compute the message digest.
- Here each of ***A, B , C, D is a 32-bit register.***
- These registers are initialized to the following values (in hexadecimal, low-order bytes first) :

word ***A: 01 23 45 67***

word *B: 89 ab* ***cd ef***

word *C:* ***fe dc ba 98***

word D: 76 **54 32 10**

STEP 4. PROCESS MESSAGE IN 16-WORD BLOCKS:

- We first define three auxiliary functions that each take as input three 32-bit words and produce **as output one 32-bit word.**

$$f(X,Y,Z) = XY \vee (\neg X)Z$$
$$g(X,Y,Z) = XY \vee XZ \vee YZ$$
$$h(X,Y,Z) = X \oplus Y \oplus Z$$

In each bit position ***f acts as a conditional: if x then y else z .***

- In each bit position *g acts* ***as a majority function: if at least two of x,y, z are one, then g has a one in*** that position.
- The function *h is the bit-wise* ***xor or parity function.***

MD 4 (MESSAGE DIGEST 4)

- **MD4 utilizes two "magic constants" in rounds two and three. The round two** constant is $\sqrt{2}$ and the round **constant is $\sqrt{3}$**

	Octal	Hex
Round 2 constant ($\sqrt{2}$):	013240474631	5A827999
Round 3 constant ($\sqrt{3}$):	015666365641	6ED9EBA1

ROUND 1

Do the following:

```
For i = 0 to N/16-1 do          /* process each 16-word block */
        Set X[j] to M[i*16+j], for j = 0, 1, ..., 15.
        Save A as AA, B as BB, C as CC, and D as DD.

        [Round 1]
          Let [A B C D i s] denote the operation
                A = (A + f(B,C,D) + X[i]) <<< s   .
          Do the following 16 operations:
                [A B C D 0  3]
                [D A B C 1  7]
                [C D A B 2  11]
                [B C D A 3  19]
                [A B C D 4  3]
                [D A B C 5  7]
                [C D A B 6  11]
                [B C D A 7  19]
                [A B C D 8  3]
                [D A B C 9  7]
                [C D A B 10 11]
                [B C D A 11 19]
                [A B C D 12 3]
                [D A B C 13 7]
                [C D A B 14 11]
                [B C D A 15 19]
```

ROUND 2

Let [A B C D i s] denote the operation

A = (A + g(B,C,D) + X[i] + 5A827999) <<< s .

Do the following 16 operations:

[A B C D 0 3]
[D A B C 4 5]
[C D A B 8 9]
[B C D A 12 13]
[A B C D 1 3]

[D A B C 5 5]
[C D A B 9 9]
[B C D A 13 13]
[A B C D 2 3]
[D A B C 6 5]
[C D A B 10 9]
[B C D A 14 13]
[A B C D 3 3]
[D A B C 7 5]
[C D A B 11 9]
[B C D A 15 13]

ROUND3

Let [A B C D i s] denote the operation

A = (A + h(B,C,D) + X[i] + 6ED9EBA1) <<< s .

Do the following 16 operations:

[A B C D 0 3]
[D A B C 8 9]
[C D A B 4 11]
[B C D A 12 15]
[A B C D 2 3]
[D A B C 10 9]
[C D A B 6 11]
[B C D A 14 15]
[A B C D 1 3]
[D A B C 9 9]
[C D A B 5 11]
[B C D A 13 15]
[A B C D 3 3]
[D A B C 11 9]
[C D A B 7 11]
[B C D A 15 15]

MD 4 (MESSAGE DIGEST 4)

Then perform the following additions:

A = A + AA

B = B + BB

C = C + CC

D = D + DD

(That is, each of the four registers is incremented by the value it had before this block was started.)

MD 4 (MESSAGE DIGEST 4)
STEP 5. OUTPUT

- The message digest produced **as output is A, B, *C, D. That is, we begin with the*** low-order byte of ***A , and end with the high-order byte of D.***

MD5

- MD5 or ”message digest 5” algorithm was designed by professor Ronald Rivest. Rivest is a professor in MIT who also invented RSA, RC5 and the MD-message digest 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 checksome 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. MD5 is almost same as MD4 but with ”safety belts”. Its 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

DESCRIPTION

- MD5 creates a 128bit message digest from data input. The output must be unique from other message digests.
- Initially designed for Digital signature.

MD5 ALGORITHM

1. Append Padding Bits
2. Append length bits
3. Initialize MD Buffer
4. Process each 512-bit block
5. Output message Digest

1. APPEND PADDING BITS

- In the first step, we add padding bits in the original message in such a way that the total length of the message is 64 bits less than the exact multiple of 512.
- Suppose we are given a message of 1000 bits. Now we have to add padding bits to the original message. Here we will add 472 padding bits to the original message. After adding the padding bits the size of the original message/output of the first step will be 1472 i.e. 64 bits less than an exact multiple of 512 (i.e. 512*3 = 1536).
- No matter the size of the message padding is always done. First a '1' bit is appended to the message and then a series of '0' bits.

FOR EXAMPLE WITH MESSAGE OF 400 BITS

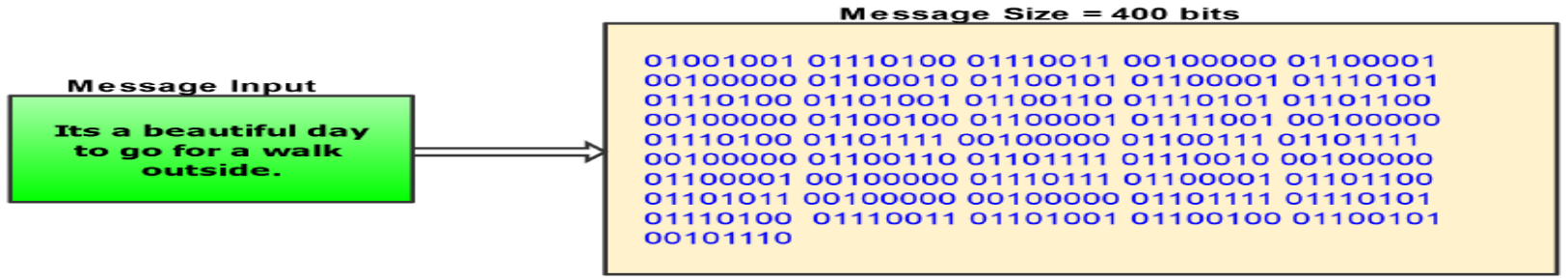


Figure 2: 400 bits original message

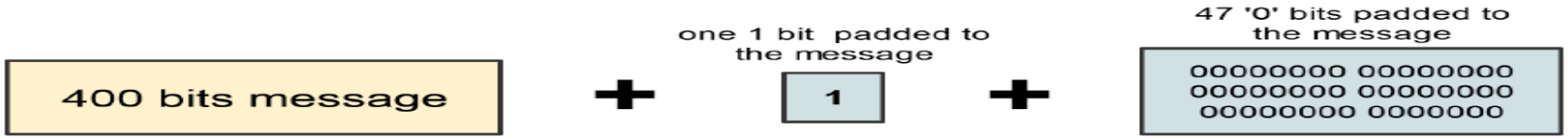


Figure 3: First one '1' bit is added and then '0' bits

FOR EXAMPLE WITH MESSAGE OF 400 BITS

- For example if our message is 400 bits then we will add one ’1’ bit and 47 ’0’ bits which gives us 448 bits which is 64 bits shy of being divisible by 512.
- If our message is 1200 bits in size than we will add one ’1’ bit and 271 ’0’ bits which gives us 1472 bits. 1472+64 is divisible by 512. At least 1 bit and at most 512 bits are padded or extended to the original message.

2. APPEND LENGTH

- The length bit is added in the output of the first step in such a way that the total number of the bits is the perfect multiple of 512. Simply, here we add the 64-bit as a length bit in the output of the first step.
- i.e. output of first step=512*n-64

  Length bits=64

After adding both we will get 512*n i.e. exact multiple of 512.

- At this point the message is divided into blocks of 512 bits each. Each 512 bits block is divided into 16 words of 32-bits each. We denote the words as M[0.....N-1] where N is a multiple of 16.

3. INITIALIZE MD BUFFER

- MD5 uses a four word buffer each 32-bits long. We denote them by A,B,C,D. These are pre-initialized as:

word A	01 23 45 67
word B	89 ab cd ef
word C	fe dc ba 98
word D	76 54 32 10

4. PROCESS EACH 512-BIT BLOCK

- This is the most important step of the MD5 algorithm.
- Here, a total of 64 operations are performed in 4 rounds.
- In the 1st round, 16 operations will be performed, 2nd round 16 operations will be performed, 3rd round 16 operations will be performed, and in the 4th round, 16 operations will be performed.
- A different function is applied on each round i.e. for the 1st round we apply the F function, for the 2nd G function, 3rd for the H function, and 4th for the I function.

F(X,Y,Z) = XY v not(X) Z

G(X,Y,Z) = XZ v Y not(Z)

H(X,Y,Z) = X xor Y xor Z

I(X,Y,Z) = Y xor (X v not(Z))

- This step uses a 64-element table T[1 ... 64] constructed from the sine function. Let T[i] denote the i-th element of the table, which is equal to the integer part of 4294967296 times abs(sin(i)), where i is in radians. The elements of the table are given in the appendix.

```
Do the following:

/* Process each 16-word block. */
For i = 0 to N/16-1 do

  /* Copy block i into X. */
  For j = 0 to 15 do
    Set X[j] to M[i*16+j].
  end /* of loop on j */

  /* Save A as AA, B as BB, C as CC, and D as DD. */
  AA = A
  BB = B


  CC = C
  DD = D
```

```
/* Round 1. */
/* Let [abcd k s i] denote the operation
     a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD  0  7  1]  [DABC  1 12  2]  [CDAB  2 17  3]  [BCDA  3 22  4]
[ABCD  4  7  5]  [DABC  5 12  6]  [CDAB  6 17  7]  [BCDA  7 22  8]
[ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
[ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]

/* Round 2. */
/* Let [abcd k s i] denote the operation
     a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
[ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
[ABCD  9  5 25]  [DABC 14  9 26]  [CDAB  3 14 27]  [BCDA  8 20 28]
[ABCD 13  5 29]  [DABC  2  9 30]  [CDAB  7 14 31]  [BCDA 12 20 32]
```

```
/* Round 3. */
/* Let [abcd k s t] denote the operation
     a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD  5  4 33]  [DABC  8 11 34]  [CDAB 11 16 35]  [BCDA 14 23 36]
[ABCD  1  4 37]  [DABC  4 11 38]  [CDAB  7 16 39]  [BCDA 10 23 40]
[ABCD 13  4 41]  [DABC  0 11 42]  [CDAB  3 16 43]  [BCDA  6 23 44]
[ABCD  9  4 45]  [DABC 12 11 46]  [CDAB 15 16 47]  [BCDA  2 23 48]

/* Round 4. */
/* Let [abcd k s t] denote the operation
     a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD  0  6 49]  [DABC  7 10 50]  [CDAB 14 15 51]  [BCDA  5 21 52]
[ABCD 12  6 53]  [DABC  3 10 54]  [CDAB 10 15 55]  [BCDA  1 21 56]
[ABCD  8  6 57]  [DABC 15 10 58]  [CDAB  6 15 59]  [BCDA 13 21 60]
[ABCD  4  6 61]  [DABC 11 10 62]  [CDAB  2 15 63]  [BCDA  9 21 64]
```

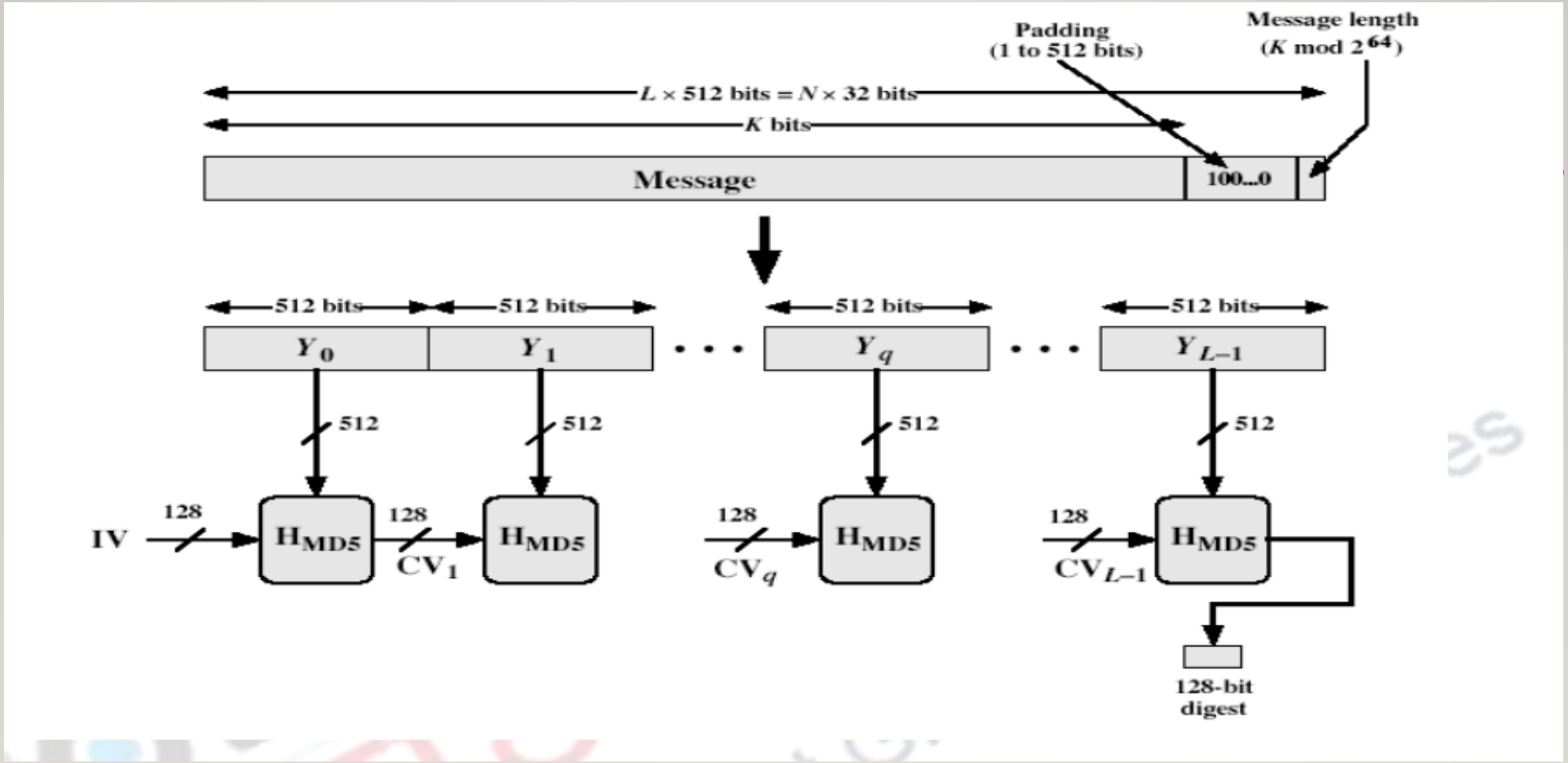
5. OUTPUT MESSAGE DIGEST

After all, rounds have been performed,

- A = A + AA
- B = B + BB
- C = C + CC
- D = D + DD

After all operations the buffer, A,B,C,D contains MD5 digest of our input message.

MD5 Overview



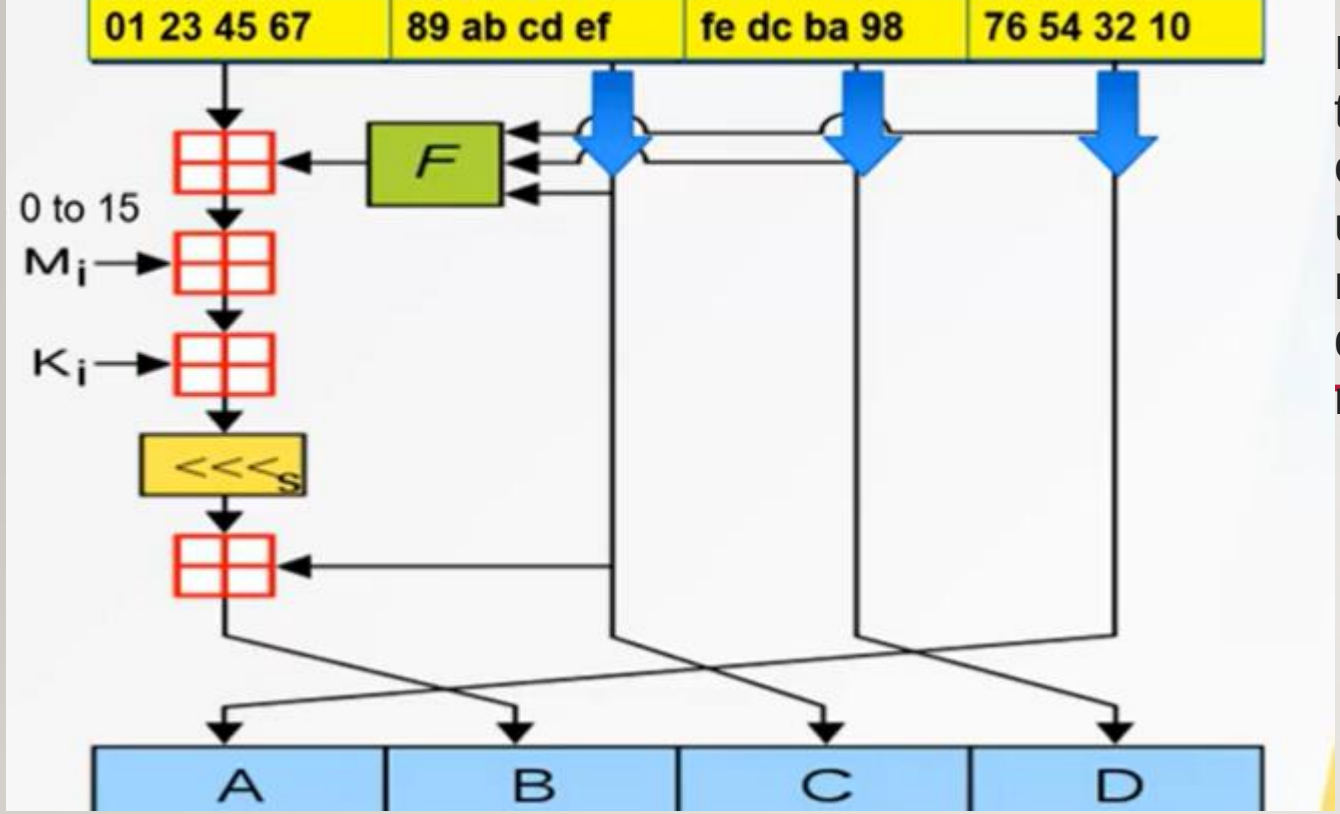
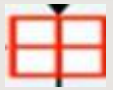


Figure 1. One MD5 operation. MD5 consists of 64 of these operations, grouped in four rounds of 16 operations. *F* is a nonlinear function; one function is used in each round. M_i denotes a 32-bit block of the message input, and K_i denotes a 32-bit constant, different for each operation. $<<<_s$ denotes a left bit rotation by *s* places; *s* varies for each operation.



denotes addition modulo 2^{32}.

The values for K are is derived from the formula:
$abs(sin(i+ 1))\times 2^{32}$

The MD5 operation is composed of
- A nonlinear operation F;
- A left-circular shift;
- Addition (modulo 2^{32}) of constants $\{T_i\}$, and
- Addition (modulo 2^{32}) of a 32-bit word $\{M_i\}$ from the message to (A, B, C, D).

64 constants.
One of these K values is used in each of the 64 operations for a 512-bit block. K1 to K16 are used in the first round, K17 to K32 are used in the second round, K33 to K48 are used in the third round, and K49 to K64 are used in the fourth round.

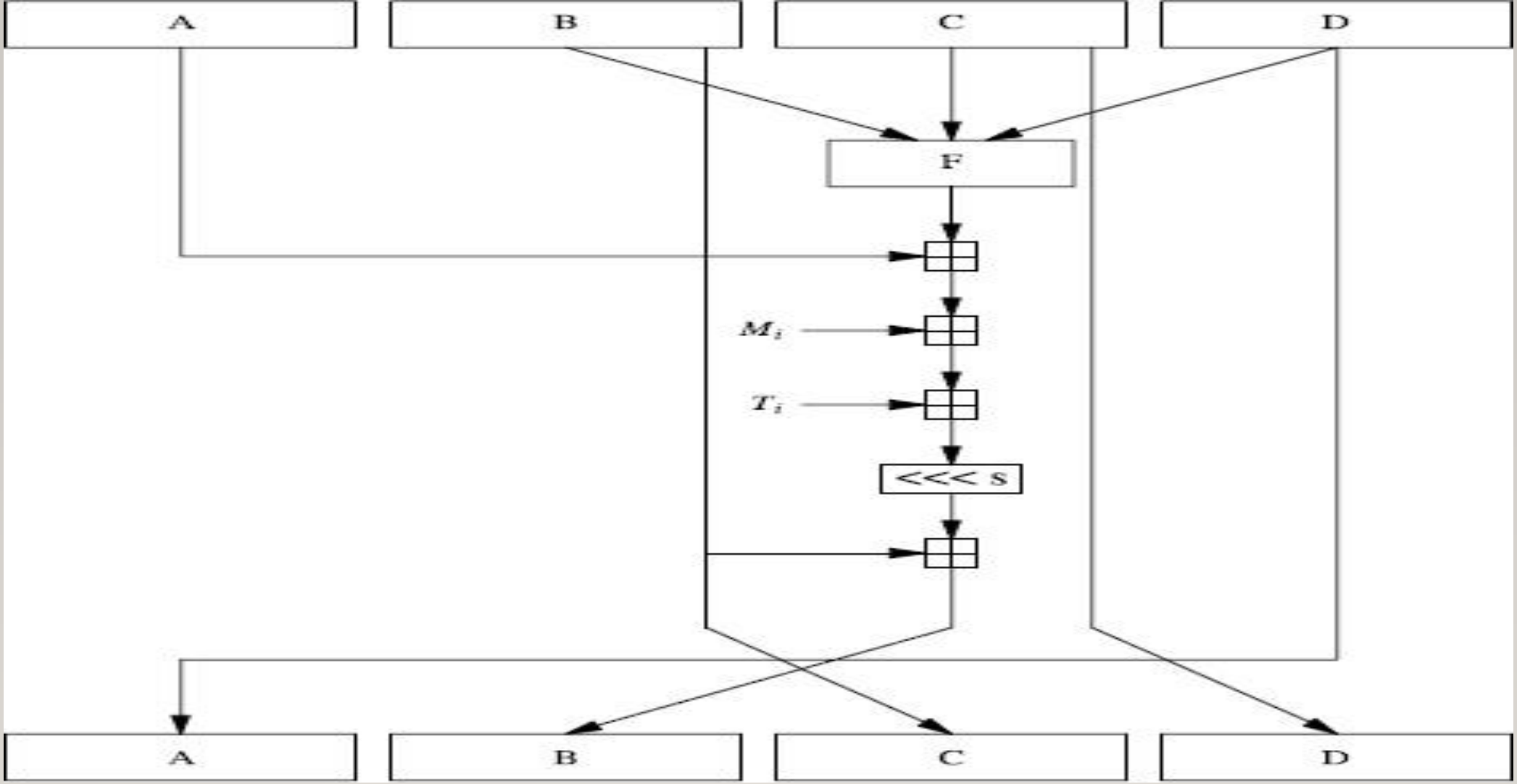
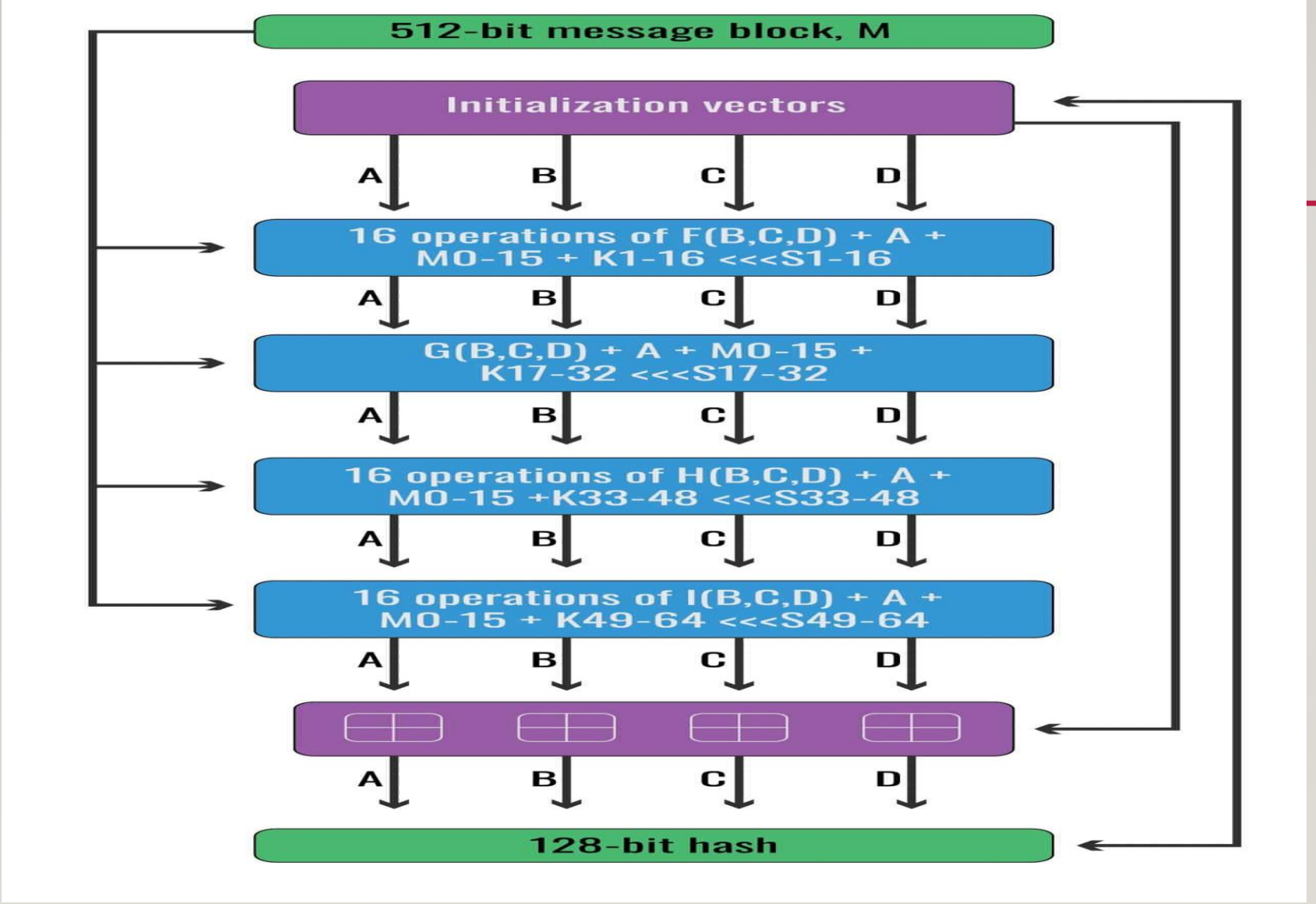


Figure Basic MD5 operation.


512-bit message block, M
Initialization vectors
A
B
C
D
16 operations of F(B,C,D) + A + M0-15 + K1-16 <<<S1-16
A
B
C
D
G(B,C,D) + A + M0-15 + K17-32 <<<S17-32
A
B
C
D
16 operations of H(B,C,D) + A + M0-15 +K33-48 <<<S33-48
A
B
C
D
16 operations of I(B,C,D) + A + M0-15 + K49-64 <<<S49-64
A
B
C
D
A
B
C
D
128-bit hash

64 CONSTANTS

K[0.. 3]	0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee
K[4.. 7]	0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501
K[8..11]	0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be
K[12..15]	0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821
K[16..19]	0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa
K[20..23]	0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8
K[24..27]	0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed
K[28..31]	0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a
K[32..35]	0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c
K[36..39]	0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70
K[40..43]	0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05
K[44..47]	0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665
K[48..51]	0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039
K[52..55]	0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1
K[56..59]	0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1
K[60..63]	0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391

DIFFERENCE BETWEEN MD4 AND MD5

S.N.	Point of Discussion	MD4	MD5
1	No of Rounds	3	4
2	Use of additive constants	Not different in all the iterations	unique additive constant in each iteration.
3	Process P in round 2		
4	The function g in round 2	(XY v XZ v YZ)	(XZ v Y not(Z))

SECURE HASH ALGORITHM

- SHA originally designed by NIST (National Institute of Standards and Technology) in 1993.
- When weaknesses were discovered in SHA (now known as SHA-0), a revised version was issued as in 1995 and is referred to as **SHA-1**.
- SHA-1 produces a hash value of 160 bits/40 hex characters /20-bytes
- In 2002, NIST produced a revised version of the standard,, that defined three new versions of SHA with hash value lengths of 256, 384, and 512 bits known as SHA-256, SHA-384, and SHA- 512, respectively. Collectively, these hash algorithms are known as **SHA-2**.
- These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1.

WORKING OF SHA1:
STEP 1: PADDING

- The first step of SHA1 is adding Padding bit to the end of original message to prepare message in multiple of 512 bits.

STEP2: APPEND THE LENGTH

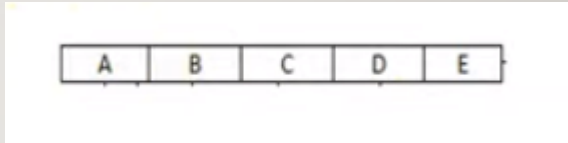
- The length of message excluding the length of padding is now calculated and appended to the end of the padding as 64-bit block. (Message length is 64 bits short of multiple of 512)

STEP3: DIVIDE THE INPUT INTO 512-BIT BLOCKS

- The input message is now divided into blocks, each of length 512 bits.

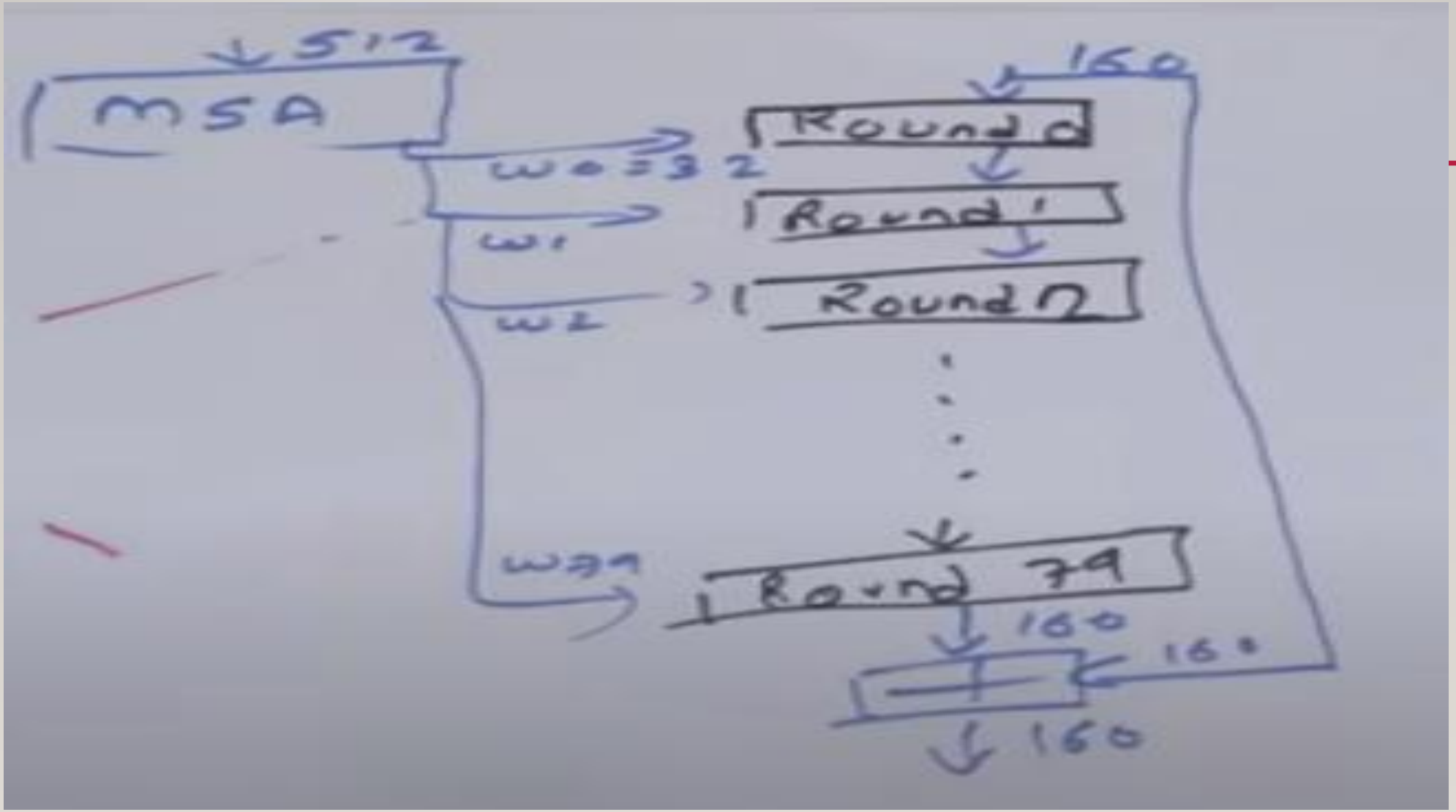
STEP4: INITIALIZE CHAINING VARIABLES

- Now five chaining (IV) variables A-E are initialized.
- Each of 32 bit variables produces 160 bits length of message digest.



STEP5: PROCESS BLOCK AND OUTPUT

- Combination of A-E chaining variable is called ABCDE, will be considered as a single register.
- Now divided the current 512-bit block into 16 sub blocks, each consisting of 32 bits. (32x16=512)
- SHA-1 has perform four rounds.
- Each round takes the current 512-bit block, the register ABCDE and constant K(t) (where t=0 to 79) as input.
- SHA consists of four rounds, each round containing 20 iteration. So total iteration is 80.
- The logical operation of a single SHA-1 iteration looks as shown in figure.


512
MSA
160
Round 0
Round 1
Round 2
Round 79
160
160
160

Processing Functions….

SHA1 requires 80 processing functions defined as:

f(t;B,C,D) = (B AND C) OR ((NOT B) AND D)	(0 <= t <= 19)
f(t;B,C,D) = B XOR C XOR D	(20 <= t <= 39)
f(t;B,C,D) = (B AND C) OR (B AND D) OR (C AND D)	(40 <= t <=59)
f(t;B,C,D) = B XOR C XOR D	(60 <= t <= 79)

- Processing Constants....

SHA1 requires 80 processing constant words defined as:

K(t) = 0x5A827999	(0 <= t <= 19)
K(t) = 0x6ED9EBA1	(20 <= t <= 39)
K(t) = 0x8F1BBCDC	(40 <= t <= 59)
K(t) = 0xCA62C1D6	(60 <= t <= 79)

PROCESSING MESSAGE…

This is the main task of SHA1 algorithm which loops through the padded and appended message in 512-bit blocks.

Input and predefined functions:

M[1, 2, ..., L]:	Blocks of the padded and appended message
f(0;B,C,D), f(1,B,C,D), ..., f(79,B,C,D)	80 Processing Functions
K(0), K(1), ..., K(79)	80 Processing Constant Words
H0, H1, H2, H3, H4, H5	5 Word buffers with initial values

SHA-1 FRAMEWORK CONTINUED

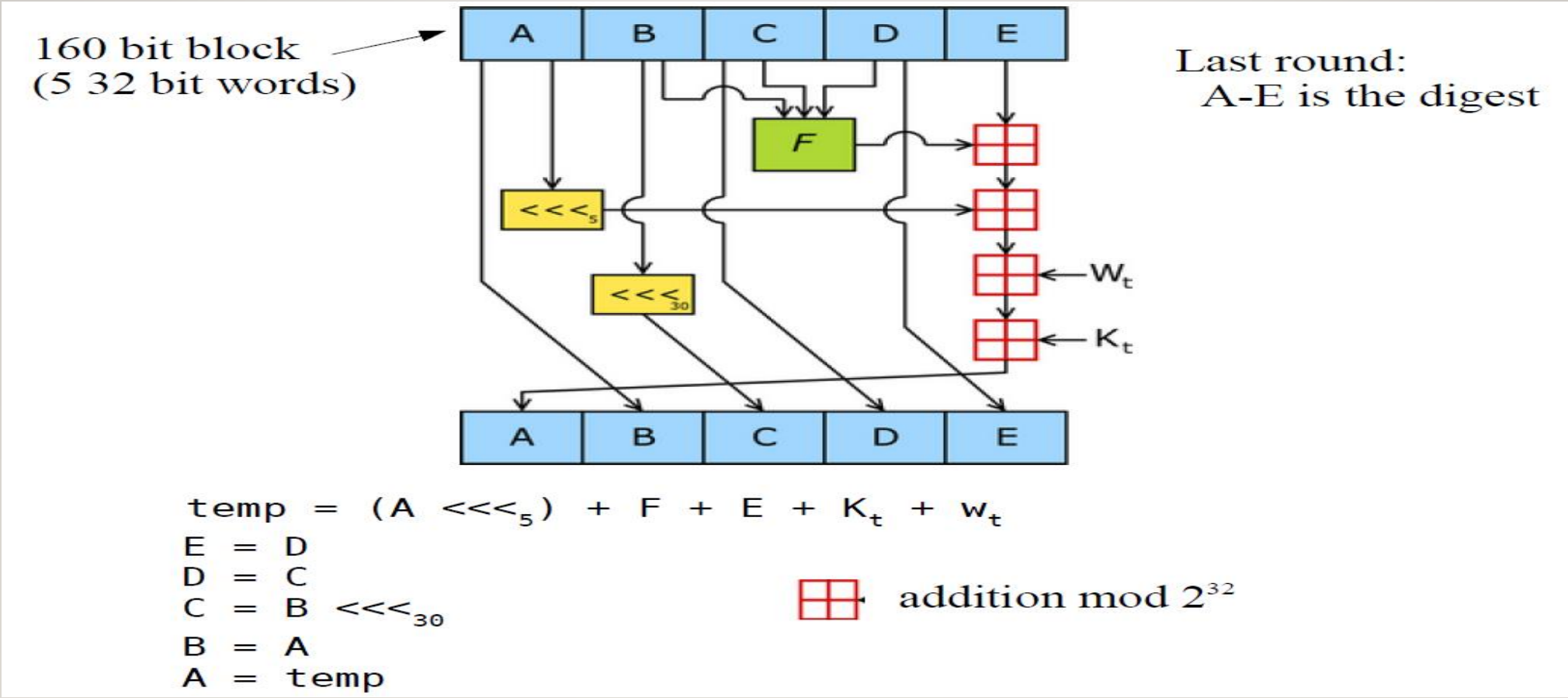
Pseudo Code….

```
For loop on k = 1 to L
        (W(0),W(1),...,W(15)) = M[k] /* Divide M[k] into 16 words */
        For t = 16 to 79 do:
            W(t) = (W(t-3) XOR W(t-8) XOR W(t-14) XOR W(t-16)) <<< 1
         A = H0, B = H1, C = H2, D = H3, E = H4
         For t = 0 to 79 do:
              TEMP = A<<<5 + f(t;B,C,D) + E + W(t) + K(t) E = D, D = C,
                                    C = B<<<30, B = A, A = TEMP
         End of for loop
        H0 = H0 + A, H1 = H1 + B, H2 = H2 + C, H3 = H3 + D, H4 = H4 + E
  End of for loop
```

Output:

H0, H1, H2, H3, H4, H5: Word buffers with final message digest

SHA1 ALGORITHM



SHA 512

- The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest.
- The input is processed in 1024-bit blocks.
- Total no of Rounds=80 (0-79)
- Each Round Consists of a constant K.
- Buffers=8
- Each Buffer Size=64 bit
- The processing consists of the following steps:

STEPS

1. Appending padding bits.
2. Append length.
3. Initialize the Buffer.
4. Process the message in 1024 bit (128-bit word blocks)
5. Output the final state values as the resulting hash.

STEP1 PADDING

- We add padding bits to original message.
- Aim is to make the length of original message is equal to a value , which is 128 bit less than an exact multiple of 1024.
- The Padding is always added even if the message is of desired length already.

SHA-512: STEP 2 APPEND LENGTH:

- A block of 128 bits is appended to the message.
- It contains the length of the original message (before the padding).
- The outcome of the first two steps yields a message that is an integer

  multiple of 1024 bits in length.
- The expanded message is represented as the sequence of 1024-bit blocks $M_1, M_2, ...M_N$, *so that the* total length of the expanded message is *N * 1024 bits*.

STEP 3 INITIALIZE HASH BUFFER:

- A 512-bit buffer is used to hold intermediate and final results of the hash function.
- The buffer can be represented as eight 64-bit registers (a, b, c, d, e, f, g, h).
- These registers are initialized to the following 64-bit integers (hexadecimal values):

a = 6A09E667F3BCC908 e = 510E527FADE682D1

b = BB67AE8584CAA73B f = 9B05688C2B3E6C1F

c = 3C6EF372FE94F82B g = 1F83D9ABFB41BD6B

d = A54FF53A5F1D36F1 h = 5BE0CD19137E2179

SHA-512

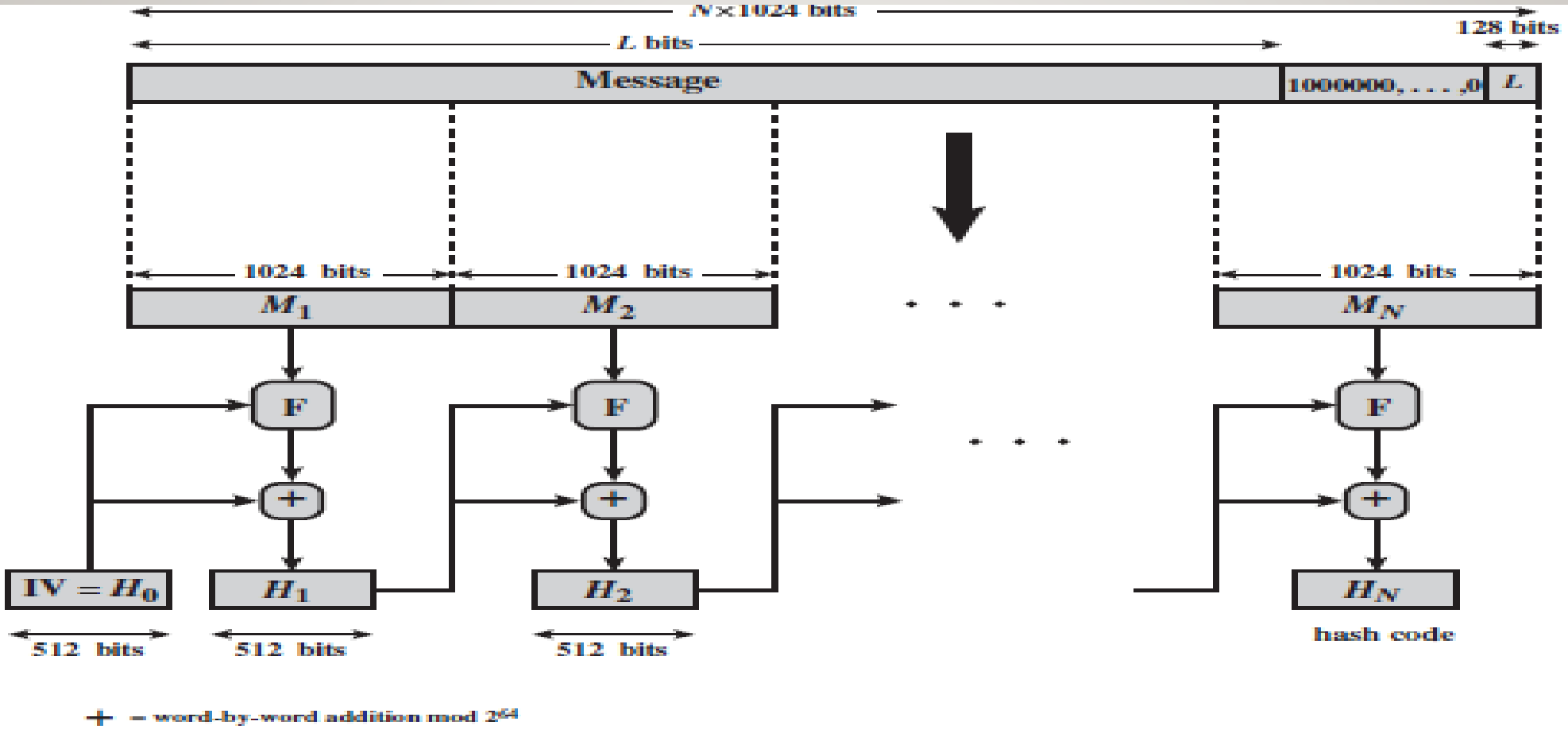


Figure 11.9 Message Digest Generation Using SHA-512

SHA-512

- **Step 4 Process message in 1024-bit (128-word) blocks:**
- Copy the above variables into corresponding lowercase variable a-f.
- Divide the current 1024 bit block into 16 sub-blocks, each subblock contains 64 bits.
- Now we have 80 rounds. In each round, We process all the 16 sub-blocks belonging to a block of 1024 bits.
- The inputs to each block are
  - All the 16 sub blocks , designated as W.
  - The variable a-h
  - Some constants designated as K.

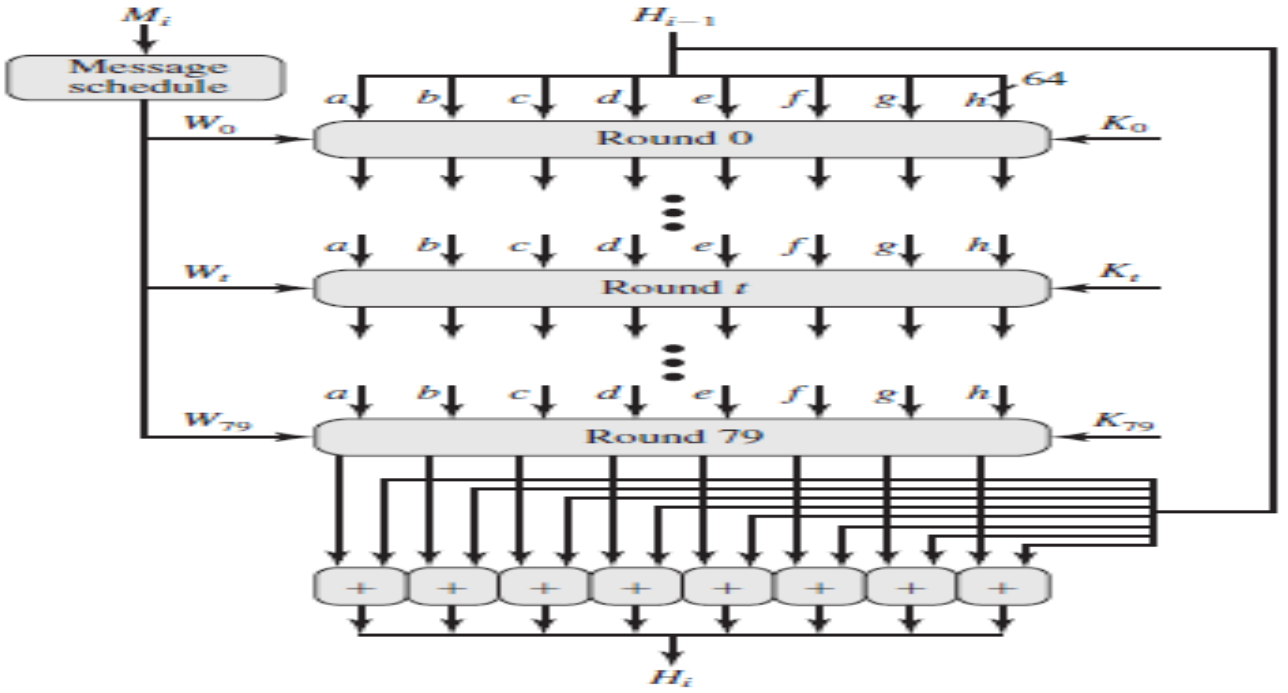


Figure 11.10 SHA-512 Processing of a Single 1024-Bit Block

SHA-512

- Each round takes as input the 512-bit buffer value, abcdefgh, and updates the contents of the buffer.
- At input to the first round, the buffer has the value of the intermediate hash value, H_{i-1}. *Each round t makes use of* a 64-bit value W_t, *derived from the current 1024-bit block being processed* (M_i).
- Each round also makes use of an additive constant K_t, *where* $0 \leq t \leq 79$ *indicates one of the 80 rounds.*
- The output of the eightieth round is added to the input to the first round (H_{i-1}) *to produce* H_i.
- The addition is done independently for each of the eight words in the buffer with each of the corresponding words in H_{i-1}, using addition modulo 2^{64}.

STEP 5 OUTPUT:

- After all *N 1024-bit blocks have been processed, the output from* the *Nth stage is the 512-bit message digest*

SHA-512

- We can summarize the behavior of SHA-512 as follows:
- $H_0 = IV$
- $H_i = SUM_{64}(H_{i-1}, abcdefgh_i)$
- $MD = H_N$
- where
- IV = initial value of the abcdefgh buffer, defined in step 3
- $abcdefgh_i$ = *the output of the last round of processing of the ith message block*
- *N = the number of blocks in the message (including padding and length fields)*
- SUM64 = addition modulo 264 performed separately on each word of the pair of inputs
- *MD = final message digest value*

SHA-512 ROUND FUNCTION:

LET US LOOK IN MORE DETAIL AT THE LOGIC IN EACH OF THE 80 STEPS OF THE PROCESSING OF ONE 512-BIT BLOCK.

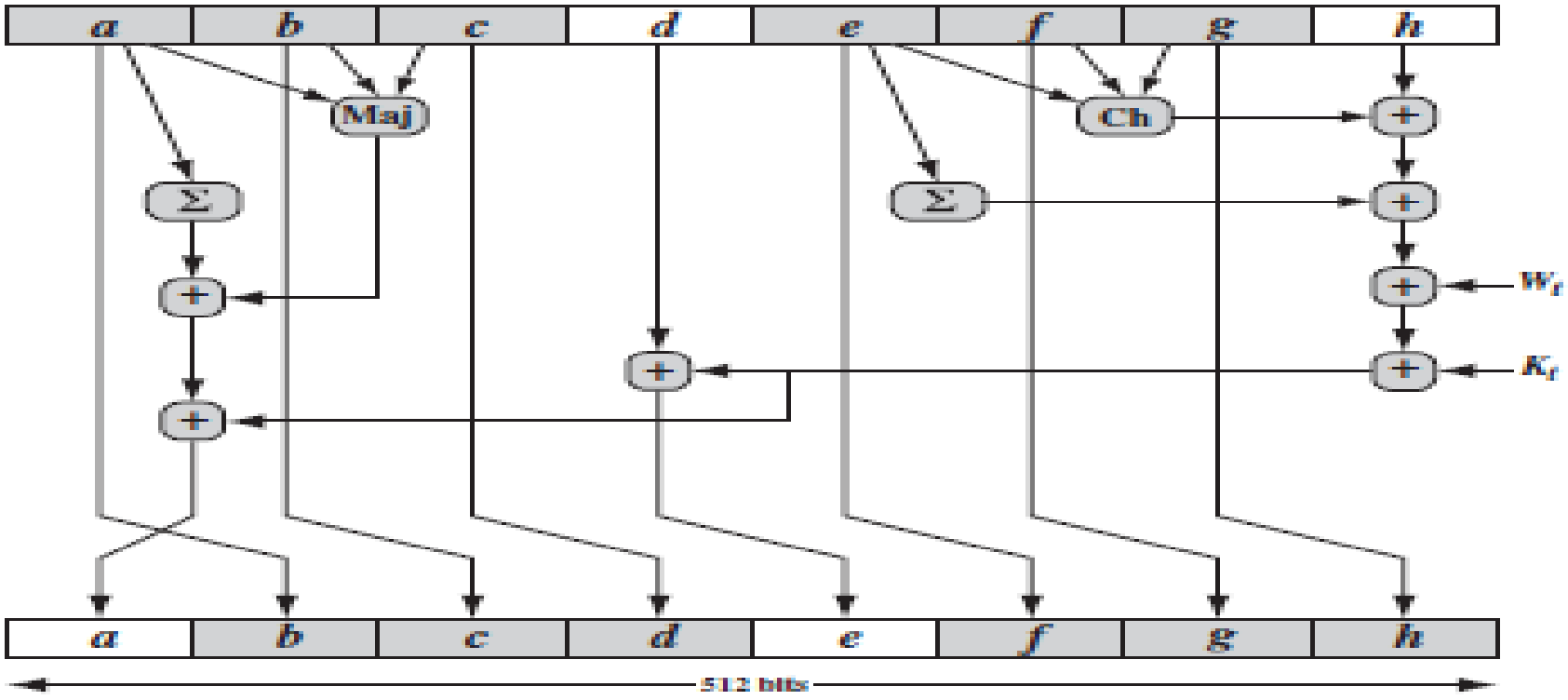


Figure 11.11 Elementary SHA-512 Operation (single round)

SHA-512

$$
\begin{aligned}
T_1 &= h + \mathrm{Ch}(e, f, g) + \left(\sum_1^{512} e\right) + W_t + K_t \\
T_2 &= \left(\sum_0^{512} a\right) + \mathrm{Maj}(a, b, c) \\
h &= g \\
g &= f \\
f &= e \\
e &= d + T_1 \\
d &= c \\
c &= b \\
b &= a \\
a &= T_1 + T_2
\end{aligned}
$$

where

t = step number; $0 \le t \le 79$

$\mathrm{Ch}(e, f, g)$ = $(e \text{ AND } f) \oplus (\text{NOT } e \text{ AND } g)$
the conditional function: If e then f else g

$\mathrm{Maj}(a, b, c)$ = $(a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c)$
the function is true only of the majority (two or three) of the arguments are true

$\left(\sum_0^{512} a\right)$ = $\mathrm{ROTR}^{28}(a) \oplus \mathrm{ROTR}^{34}(a) \oplus \mathrm{ROTR}^{39}(a)$

$\left(\sum_1^{512} e\right)$ = $\mathrm{ROTR}^{14}(e) \oplus \mathrm{ROTR}^{18}(e) \oplus \mathrm{ROTR}^{41}(e)$

$\mathrm{ROTR}^n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits

SHA-512

- W_t = *a 64-bit word derived from the current 1024-bit input block*
- K_t = *a 64-bit additive constant*
- + = addition modulo 2^{64}
- Two observations can be made about the round function.
- Six of the eight words of the output of the round function involve simply permutation (*b, c, d, f , g, h) by means of rotation.This is indicated by shading in* Figure 11.11.
- Only two of the output words (*a, e) are generated by substitution.Word* e *is a* function of input variables (*d,* e, *f , g, h), as well as the round word* W_t *and the* constant K_t.*Word a is a function of all of the input variables except d, as well* as the round word W_t *and the constant* K_t.
- It remains to indicate how the 64-bit word values W_t *are derived from the*

  1024-bit message.
- The first 16 values of *Wt are* taken directly from the 16 words of the current block.

SHA-512

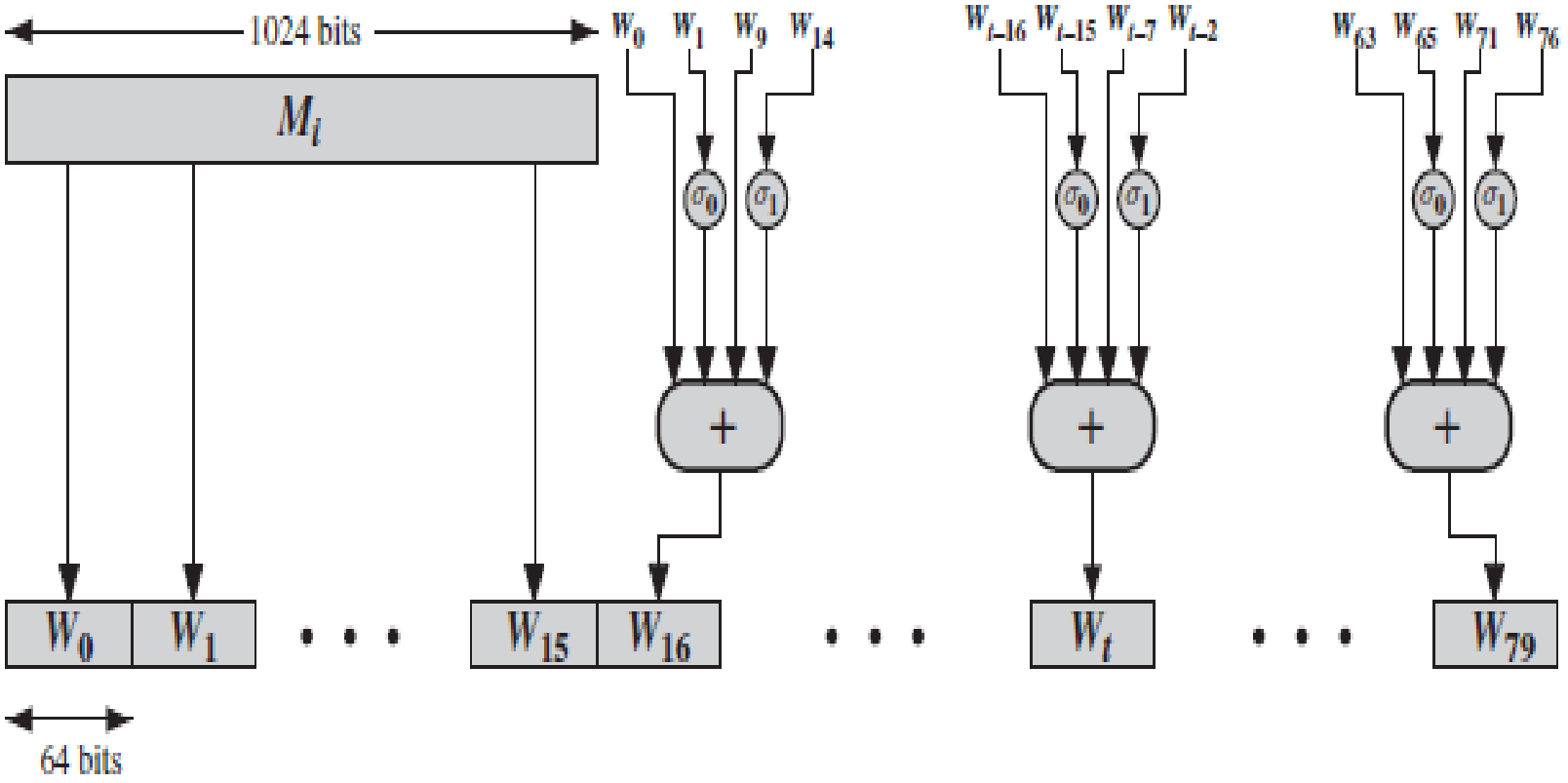


Figure 11.12 Creation of 80-word Input Sequence for SHA-512 Processing of Single Block

SHA-512

- The remaining values are defined as:

$$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$$

where

$$\sigma_0^{512}(x) = \text{ROTR}^1(x) \oplus \text{ROTR}^8(x) \oplus \text{SHR}^7(x)$$
$$\sigma_1^{512}(x) = \text{ROTR}^{19}(x) \oplus \text{ROTR}^{61}(x) \oplus \text{SHR}^6(x)$$
$$\text{ROTR}^n(x) = \text{circular right shift (rotation) of the 64-bit argument } x \text{ by } n \text{ bits}$$
$$\text{SHR}^n(x) = \text{left shift of the 64-bit argument } x \text{ by } n \text{ bits with padding by zeros on the right}$$
$$+ = \text{addition modulo } 2^{64}$$

- In the first 16 steps of processing, the value of W_t *is equal to the corresponding* word in the message block.
- For the remaining 64 steps, the value of W_t *consists of the circular left shift by one bit of the XOR of four of the preceding* values of W_t *with two of those values subjected to shift and rotate operations.*