# Hash Algorithms

# Hash Algorithms

- see similarities in the evolution of hash functions & block ciphers
  - increasing power of brute-force attacks
  - leading to evolution in algorithms
  - from DES to AES in block ciphers
  - from MD4 & MD5 to SHA-1 & RIPEMD-160 in hash algorithms
- likewise tend to use common iterative structure as do block ciphers

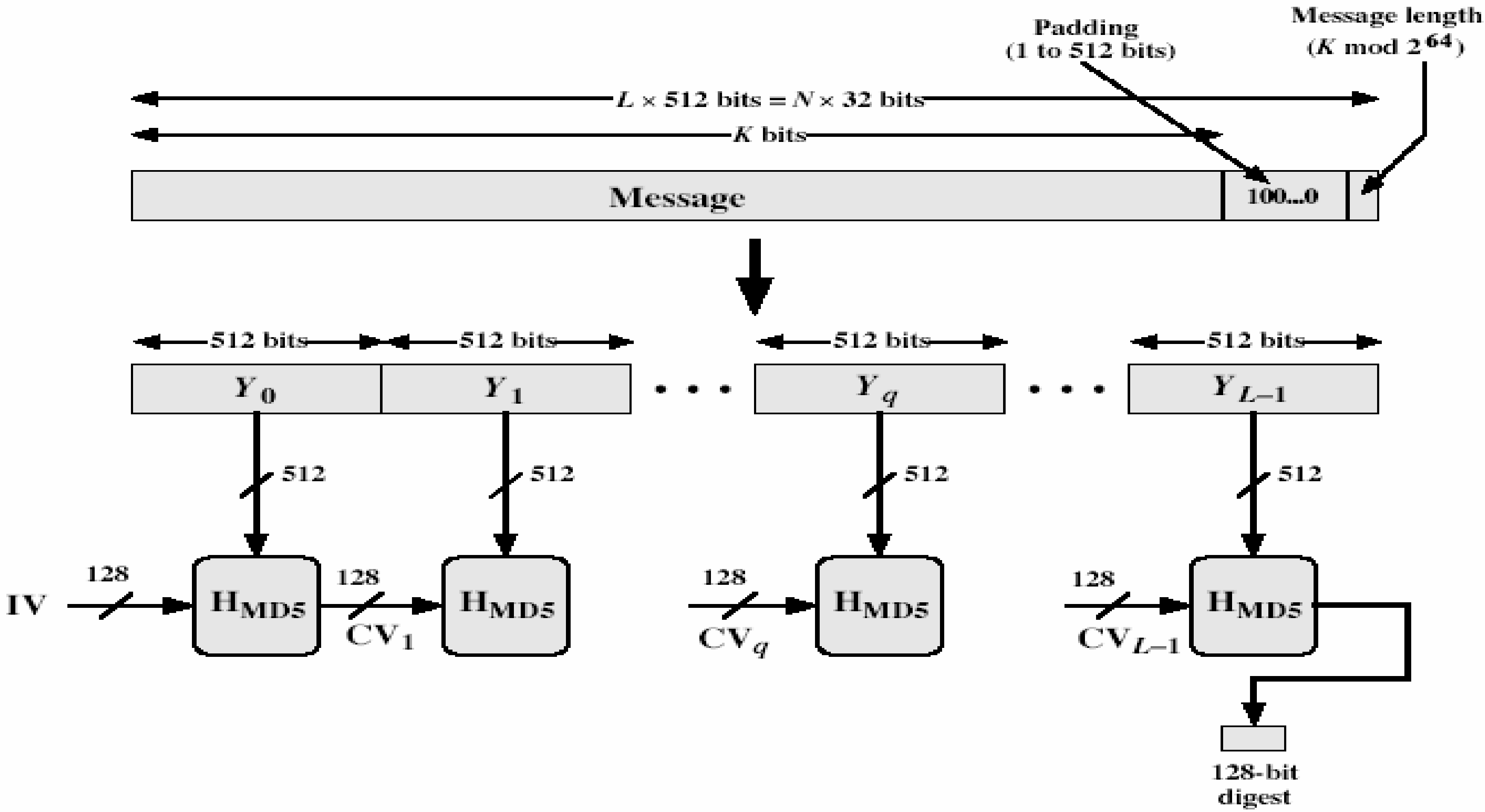
# MD5

- designed by Ronald Rivest (the R in RSA)
- latest in a series of MD2, MD4
- produces a 128-bit hash value
- until recently was the most widely used hash algorithm
  - in recent times have both brute-force & cryptanalytic concerns
- specified as Internet standard RFC1321

# MD5 Overview

1. pad message so its length is 448 mod 512
2. append a 64-bit length value to message
3. initialise 4-word (128-bit) MD buffer (A,B,C,D)
4. process message in 16-word (512-bit) blocks:
   - using 4 rounds of 16 bit operations on message block & buffer
   - add output to buffer input to form new buffer value
5. output hash value is the final buffer value

# MD5 Overview



# Implementation Steps

## Step1 Append padding bits

- The input message is "padded" (extended) so that its length (in bits) equals to 448 mod 512.
- Padding is always performed, even if the length of the message is already 448 mod 512.
- Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448 mod 512.
- At least one bit and at most 512 bits are appended.

# Implementation Steps

## Step2. Append length

- A 64-bit representation of the length of the message is appended to the result of step1.
- If the length of the message is greater than 2^64, only the low-order 64 bits will be used.
- The resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits.
- The input message will have a length that is an exact multiple of 16 (32-bit) words.

# Implementation Steps

## Step3. Initialize MD buffer

- A four-word buffer (A, B, C, D) is used to compute the message digest.  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

# Implementation Steps

## Step4. Process message in 16-word blocks

- Four functions will be defined such that each function takes an input of three 32-bit words and produces a 32-bit word output.

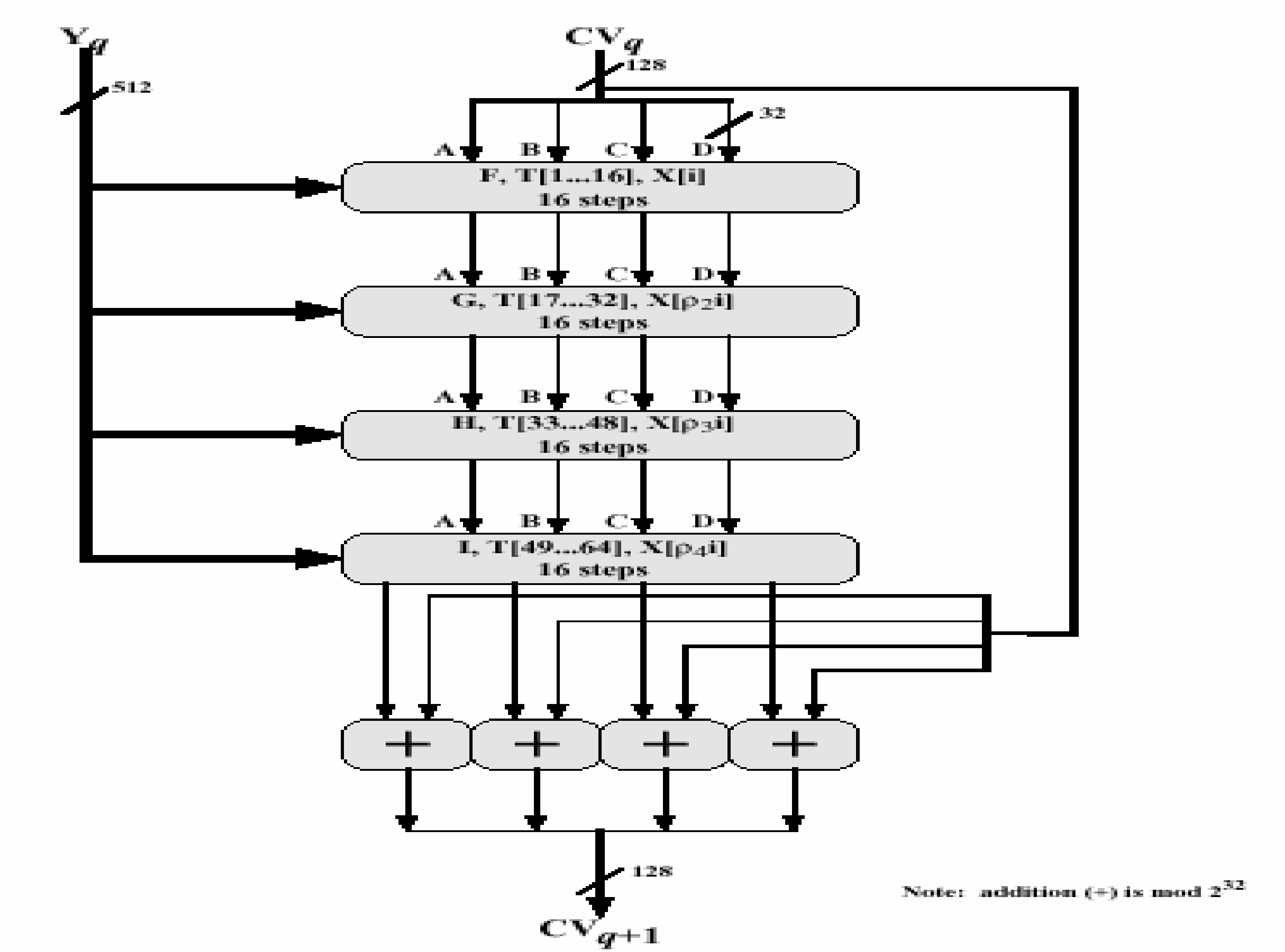
F (X, Y, Z) = XY or not (X) Z

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

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

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

# 4 Rounds



Yq
512
CVq
128
32
A B C D
F, T[1...16], X[i]
16 steps
A B C D
G, T[17...32], X[ρ2i]
16 steps
A B C D
H, T[33...48], X[ρ3i]
16 steps
A B C D
I, T[49...64], X[ρ4i]
16 steps
+ + + +
128
CVq+1
Note: addition (+) is mod 2^32

# Implementation Steps

Step 5. Output:

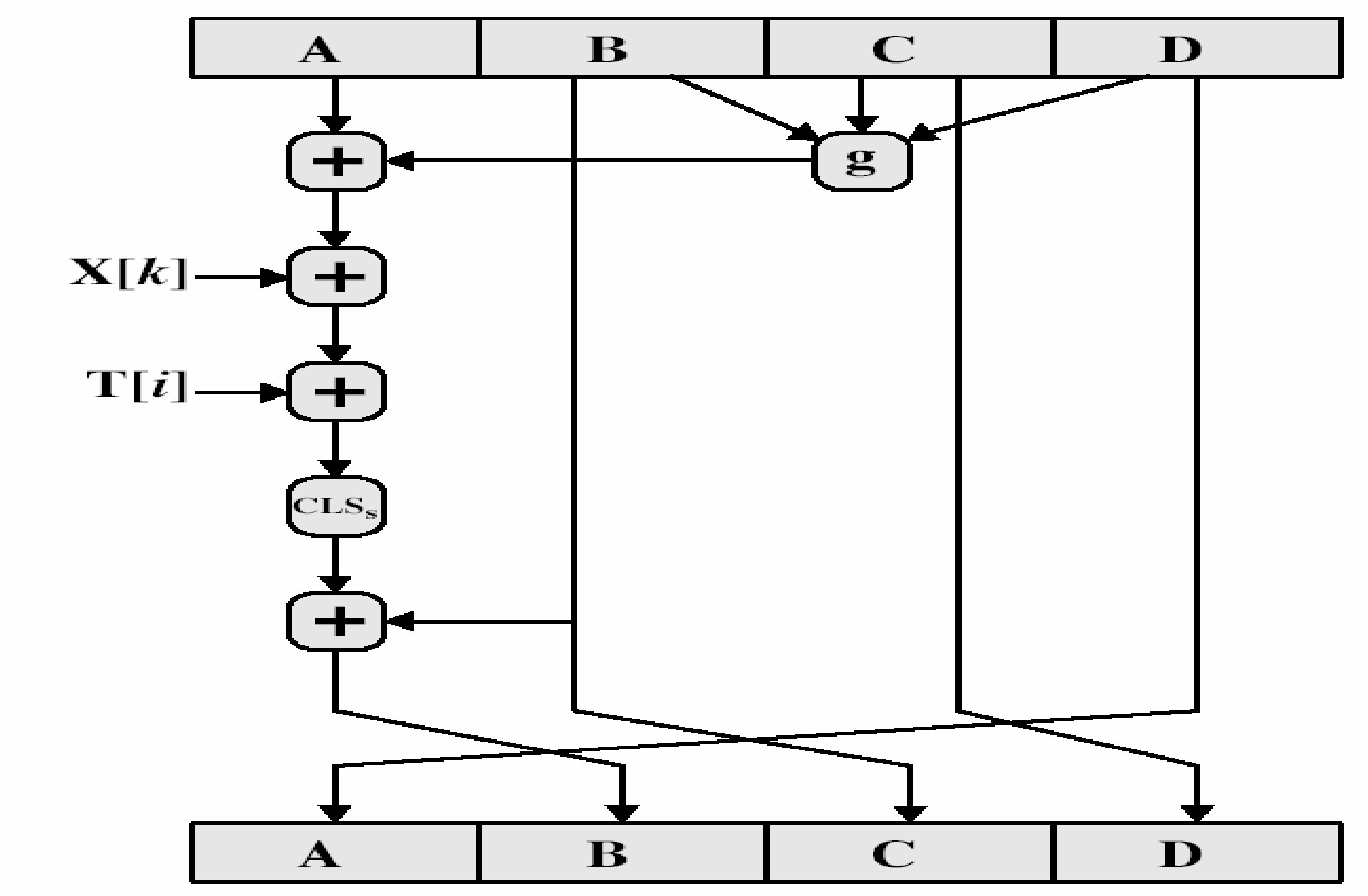
- The message digest produced as output is A, B, C, D.
- That is, we begin with the low-order byte of A, and end with the high-order byte of D.

# MD5 Compression Function

- each round has 16 steps of the form:

  `a = b+((a+g(b,c,d)+X[k]+T[i])<<<s)`
- a,b,c,d refer to the 4 words of the buffer, but used in varying permutations
  - note this updates 1 word only of the buffer
  - after 16 steps each word is updated 4 times
- where g(b,c,d) is a different nonlinear function in each round (F,G,H,I)
- T[i] is a constant value derived from sin

# MD5 Compression Function



# Table T, constructed from the sine function

- 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 following slide.

# Table T, constructed from the sine function

| | | | |
|---|---|---|---|
| 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] = 699D6122 | 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] = 02441453 | 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] = 04881D05 | 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 |

# MD4

- precursor to MD5
- also produces a 128-bit hash of message
- has 3 rounds of 16 steps vs 4 in MD5
- design goals:
  - collision resistant (hard to find collisions)
  - direct security (no dependence on "hard" problems)
  - fast, simple, compact
  - favours little-endian systems (eg PCs)

# Strength of MD5

- MD5 hash is dependent on all message bits
- Rivest claims security is good as can be
- known attacks are:
  - Berson 92 attacked any 1 round using differential cryptanalysis (but can’t extend)
  - Boer & Bosselaers 93 found a pseudo collision (again unable to extend)
  - Dobbertin 96 created collisions on MD compression function (but initial constants prevent exploit)
- conclusion is that MD5 looks vulnerable soon

# Secure Hash Algorithm (SHA-1)

- SHA was designed by NIST & NSA in 1993, revised 1995 as SHA-1
- US standard for use with DSA signature scheme
  - standard is FIPS 180-1 1995, also Internet RFC3174
  - nb. the algorithm is SHA, the standard is SHS
- produces 160-bit hash values
- now the generally preferred hash algorithm
- based on design of MD4 with key differences

# SHA Overview

1. pad message so its length is 448 mod 512
2. append a 64-bit length value to message
3. initialise 5-word (160-bit) buffer (A,B,C,D,E) to (67452301,efcdab89,98badcfe,10325476,c3d2e1f0)
4. process message in 16-word (512-bit) chunks:
   - expand 16 words into 80 words by mixing & shifting
   - use 4 rounds of 20 bit operations on message block & buffer
   - add output to input to form new buffer value
5. output hash value is the final buffer value

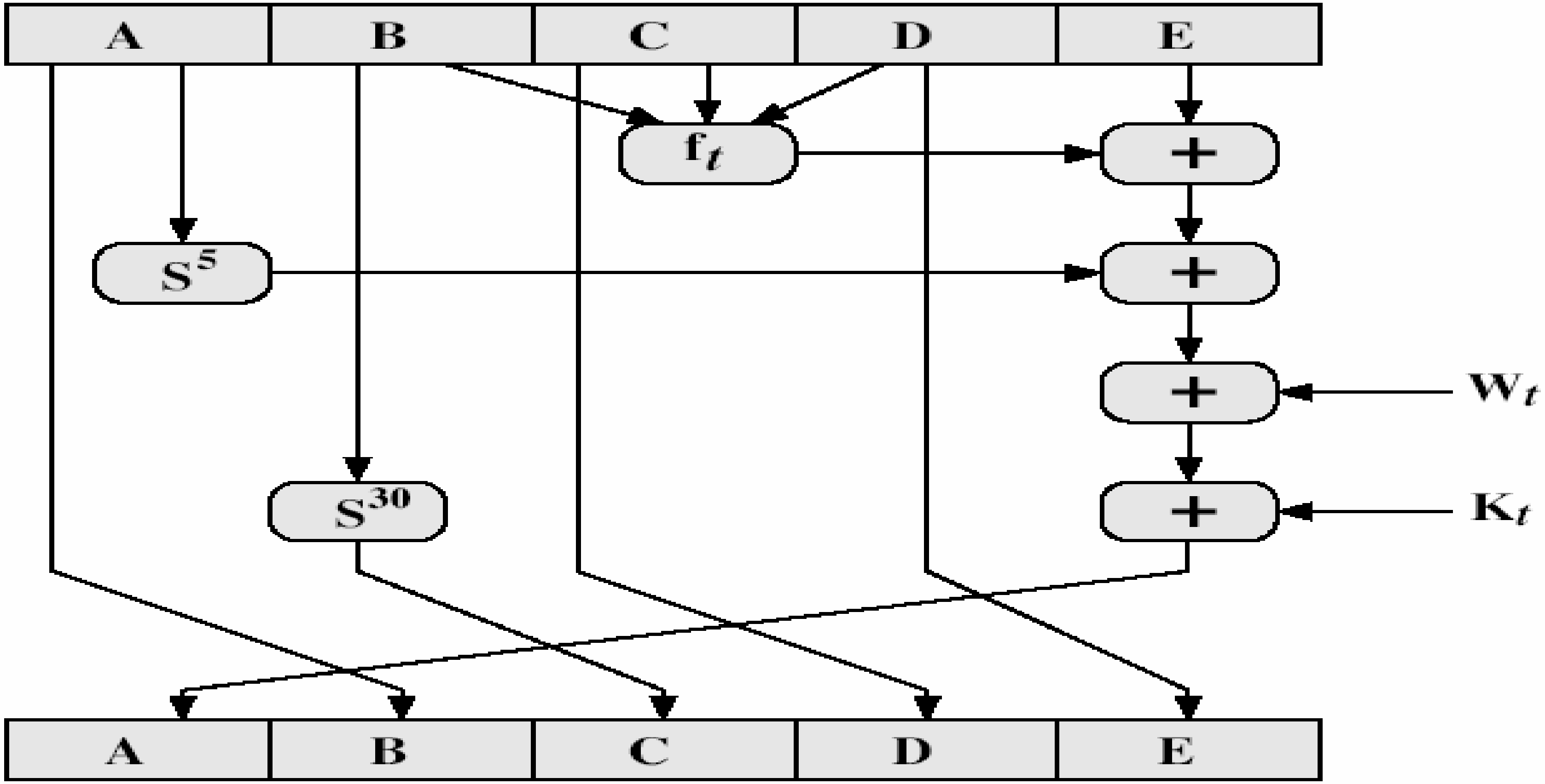
# SHA-1 Compression Function

- each round has 20 steps which replaces the 5 buffer words thus:

```
(A,B,C,D,E) <-
   (E+f(t,B,C,D)+(A<<5)+Wt+Kt),A,(B<<30),C,D)
```

- a,b,c,d refer to the 4 words of the buffer
- t is the step number
- `f(t,B,C,D)` is nonlinear function for round
- $W_t$ is derived from the message block
- $K_t$ is a constant value derived from sin

# SHA-1 Compression Function



# SHA-1 verses MD5

- brute force attack is harder (160 vs 128 bits for MD5)
- not vulnerable to any known attacks (compared to MD4/5)
- a little slower than MD5 (80 vs 64 steps)
- both designed as simple and compact
- optimised for big endian CPU's (vs MD5 which is optimised for little endian CPU's)

# Revised Secure Hash Standard

- NIST have issued a revision FIPS 180-2
- adds 3 additional hash algorithms
- SHA-256, SHA-384, SHA-512
- designed for compatibility with increased security provided by the AES cipher
- structure & detail is similar to SHA-1
- hence analysis should be similar