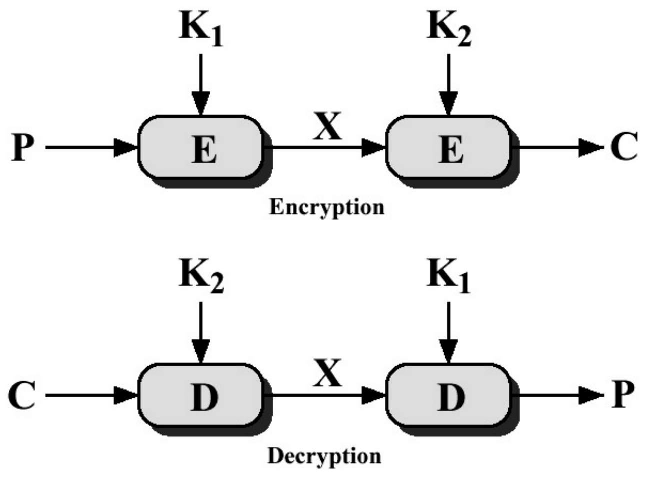
# Triple DES

- DES is vulnerable to brute-force attacks
- Double DES –
  - Two encryption stages and two keys
  - Input – plaintext P, two encryptions $K_1$, $K_2$
  - Output – ciphertext C
    $$C = E_{k2}[E_{k1}[P]]$$
  - Decryption –
    $$P = D_{k1}[D_{k2}[C]]$$
  - Meet-in-the-Middle Attack
    - Since $C = E_{k2}[E_{k1}[P]] \Rightarrow X = E_{k1}[P] = D_{k2}[C]$
    - Given a known pair (P, C), the attack proceeds as follows
      - encrypt P for all $2^{56}$ possible values of $K_1$, store the results in a table and then sort the table by the values of X



# Triple DES (cont.)

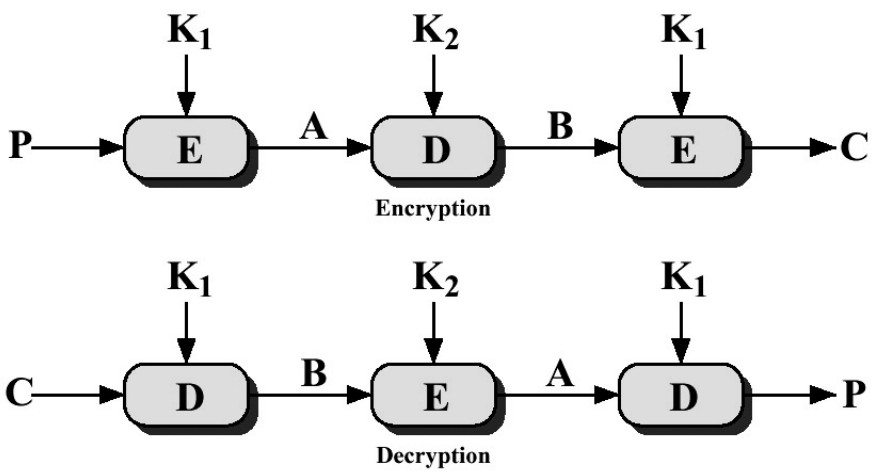
- Meet-in-the-Middle Attack (cont.)
  - encrypt P for all $2^{56}$ possible values of $K_1$, store the results in a table and then sort the table by the values of X
  - Decrypt C using all $2^{56}$ possible values of $K_2$, check the result against the table for a match
    - If matches, take the two keys against a new known (P, C) pair, if correct ciphertext is produced, accept them as correct keys
- In double DES, there are $2^{112}$ possible keys; and for a given P, there are $2^{64}$ possible values of C could be produced
- On average, for a given plaintext P, the number of different 112-bit keys that will produce a given ciphertext C is $2^{112}/2^{64} = 2^{48}$
- $2^{48}$ false alarms on the first (P, C) pair
- For an additional (P, C) pair, the false alarm rate is reduced to $2^{48}/2^{64} = 2^{-16}$
- The probability of finding the correct keys is $1 – 2^{-16}$

# Triple DES (cont.)

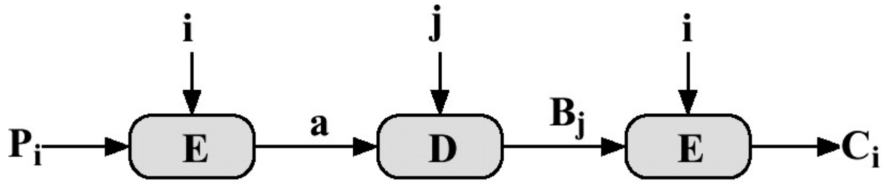
## Triple DES with Two Keys –

- To counter the meet-in-the-middle attack, a method is to use three stages of encryption with three different keys
  - The cost of known-plaintext attack is $2^{112}$, and the key size is $56 \times 3 = 168$ bits
  - Tuchman proposed a triple encryption method using two keys in an order of encrypt-decrypt-encrypt sequence

$$C = E_{K1}[D_{K2}[E_{K1}[P]]]$$



# Triple DES (cont.)




## Triple DES with Two Keys (cont.)

- The cost of a brute-force key search is $2^{112} \approx (5\times10^{33})$
- The cost of differential cryptanalysis exceeds $10^{52}$
- Known-plaintext cryptanalysis:
  - Obtain *n* (P, C) pairs and place these in Table 1 sorted on P
  - Arbitrarily select a value *a* for A, for each of the $2^{56}$ possible keys $K_1 = i$, calculate $P_i$ that produces *a*:

    $P_i = D_i[a]$

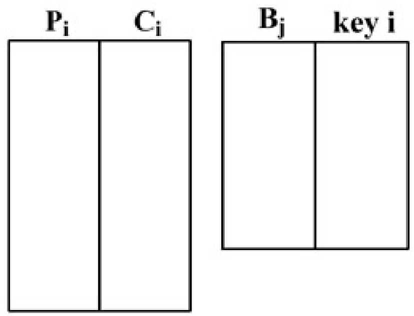
    For each match $P_i$, create an entry in Table 2 consisting of $K_1$ and B:

    $B = D_i[C]$

    Sort Table 2 on the value of B
  - for each possible $K_2 = j$, calculate $2^{nd}$ intermediate value:

    $B_j = D_j[a]$

    if $B_j$ is in Table 2, then $(i, j)$ is a possible candidate for $(K_1, K_2)$



# Triple DES (cont.)

## Triple DES with Two Keys (cont.)

### Known-plaintext cryptanalysis (cont.)

- For a known (P, C), the probability of success for a single value of *a* is $1/2^{64}$
- Given *n* (P, C) pairs, the probability of success for a single value of *a* is $n/2^{64}$
- For large n, the expected number of values of a must be tried is

$$\frac{2^{64}+1}{n+1} \approx \frac{2^{64}}{n}$$

- The expected running time of the attack is on the order of

$$\left(2^{56}\right)\frac{2^{64}}{n} = 2^{120-\log_2 n}$$

# Triple DES (cont.)

## Triple DES with Three Keys –

- Key length is 168 bits:

  $$C = E_{K3}[D_{K2}[E_{K1}[P]]]$$

- Backward compatibility with DES is provided by putting $K_3 = K_2$ or $K_1 = K_2$

# International Data Encryption Algorithm (IDEA)

- **Design Principles – 128-bit key, 64-bit block of plaintext**
  - Cryptographic strength
    - Block length
    - Key length
    - Confusion
    - Diffusion
  - Implementation considerations
    - Design principles for software implementation:
      - Use subblocks – IDEA uses16-bit subblocks
      - Use simple operations – XOR, integer addition and multiplication
    - Design principles for hardware implementation:
      - Similarity of encryption and decryption
      - Regular structure

# IDEA (cont.)

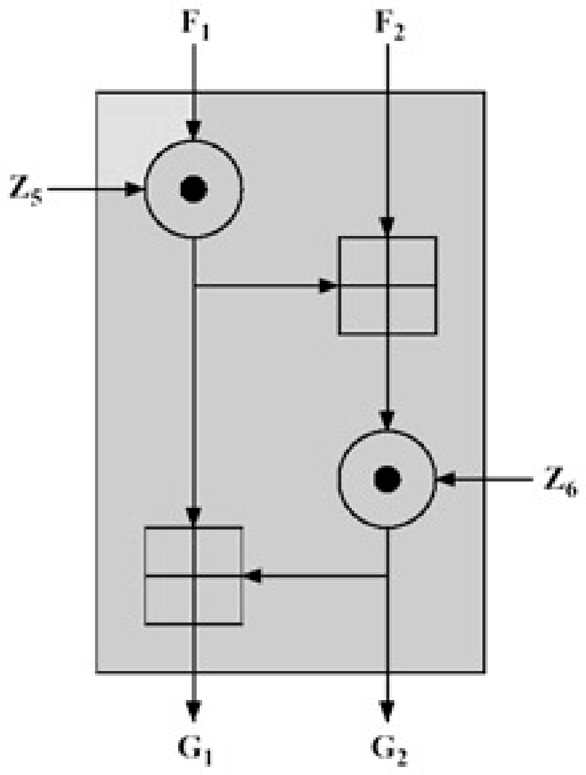
## Design Principles (cont.)

- In IDEA, confusion is achieved by mixing three different operations (16-bit input and 16-bit output)
  - Bit-by-bit XOR, denoted as $\oplus$
  - Addition of integers modulo $2^{16}$ (modulo 65536), denoted as $\boxplus$
  - Multiplication of integers modulo $2^{16}+1$ (modulo 65537), denoted as $\odot$
    - A block of all 0's is treated as representing $2^{16}$
    - For example, 0000000000000000 $\odot$ 1000000000000000 = 1000000000000001 (since $2^{16} \times 2^{15} \bmod (2^{16} + 1) = 2^{15} + 1$)
  - These three operations are incompatible in the sense that
    - No pair of the three operations satisfies a distributive law $a \boxplus (b \odot c) \neq (a \boxplus b) \odot (a \boxplus c)$
    - No pair of the three operations satisfies an associative law $a \boxplus (b \oplus c) \neq (a \boxplus b) \oplus c$

# IDEA (cont.)

## Design Principles (cont.)

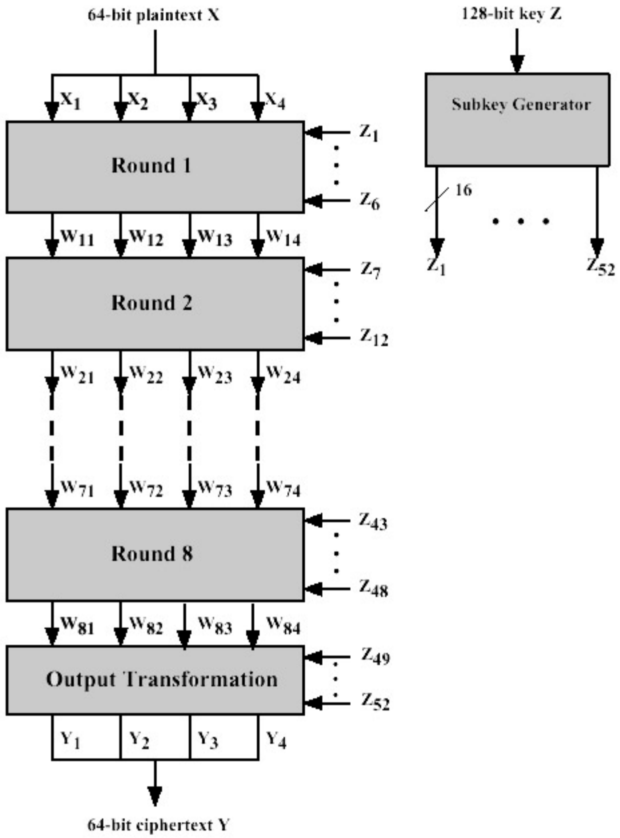
- In IDEA, diffusion is provided by the basic building block of the algorithm – the multiplication/addition (MA) structure
  - Input – 2 16-bit plaintext ($F_1$ and $F_2$)
    2 16-bit subkeys ($Z_5$ and $Z_6$)
  - Output – 2 16-bit output ($G_1$ and $G_2$)
  - Each output bit of the first round depends on every bit of the plaintext and on every bit of the subkeys
  - This structure is repeated 8 times to provide effective diffusion



# IDEA (cont.)

## IDEA Encryption –

- Input – 64-bit plaintext, 128-bit key
- Output – 64-bit ciphertext
- Encryption algorithm consists of 8 rounds followed by a final transformation function
- Round function –
  - Input – 4 16-bit subblocks, 6 16-bit subkeys
  - Output – 4 16-bit subblocks
- Output transformation function –
  - Input – 4 16-bit subblocks, 4 16-bit subkeys
  - Output – 4 16-bit subblocks
- Subkey generator –
  - Input – 128-bit key
  - Output – 52 16-bit subkeys

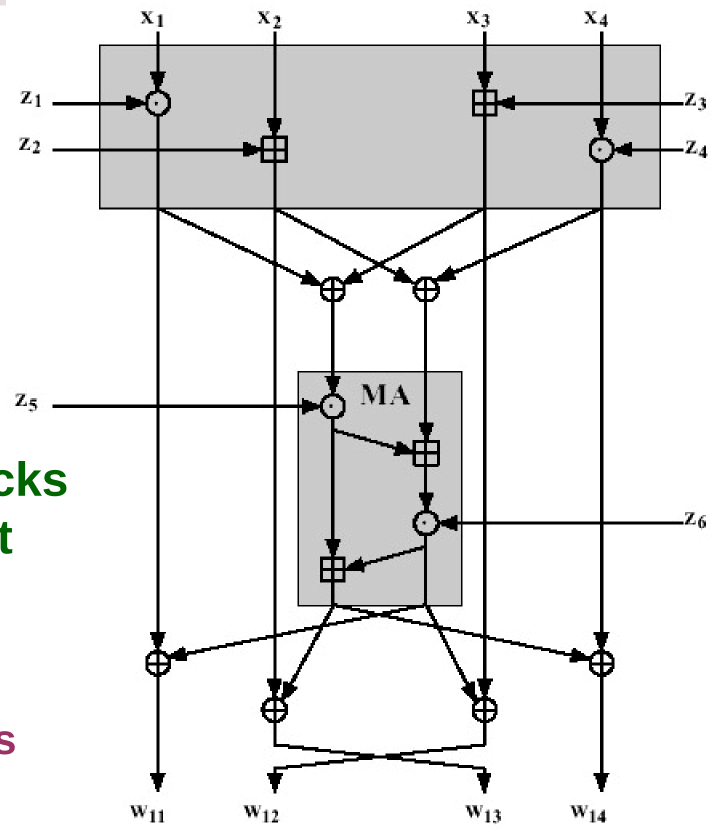


# IDEA (cont.)

## IDEA Encryption (cont.)

- Details of a Single Round

1. **Transformation – use addition and multiplication operations**
   - Input – 4 subblocks ($X_1$, $X_2$, $X_3$, $X_4$) and 4 subkeys ($Z_1$, $Z_2$, $Z_3$, $Z_4$)
2. **XOR operation – The 4 output subblocks are XORed to form 2 16-bit blocks that are inputs to the MA structure**
3. **MA structure –**
   - Input – 2 16-bit blocks, 2 16-bit subkeys
   - Output – 2 16-bit output blocks
4. **XOR operation – The 4 outputs from the upper transformation are XORed with the 2 outputs of the MA structure to produce 4 outputs**

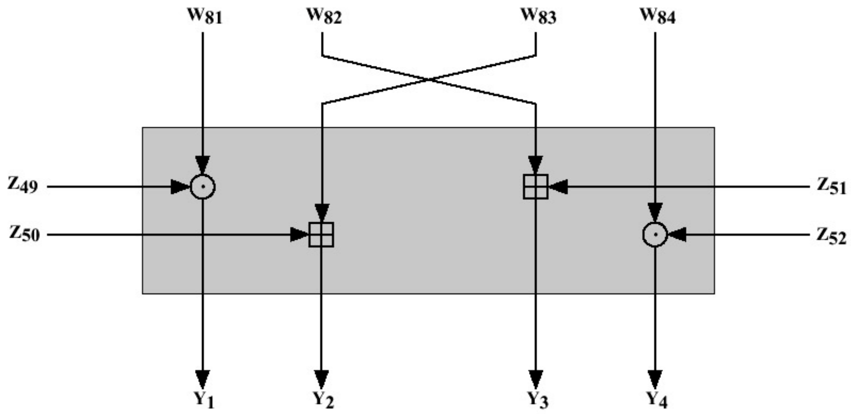


# IDEA (cont.)

## IDEA Encryption (cont.)

### Output Transformation Function

- Input – 4 16-bit blocks, 4 16-bit subkeys
- Output – 4 16-bit output blocks
- Similar to the upper transformation of a single round
- The 2nd and 3rd inputs are interchanged such that decryption has the same structure as encryption



# IDEA (cont.)

## IDEA Encryption (cont.)

### Subkey Generation

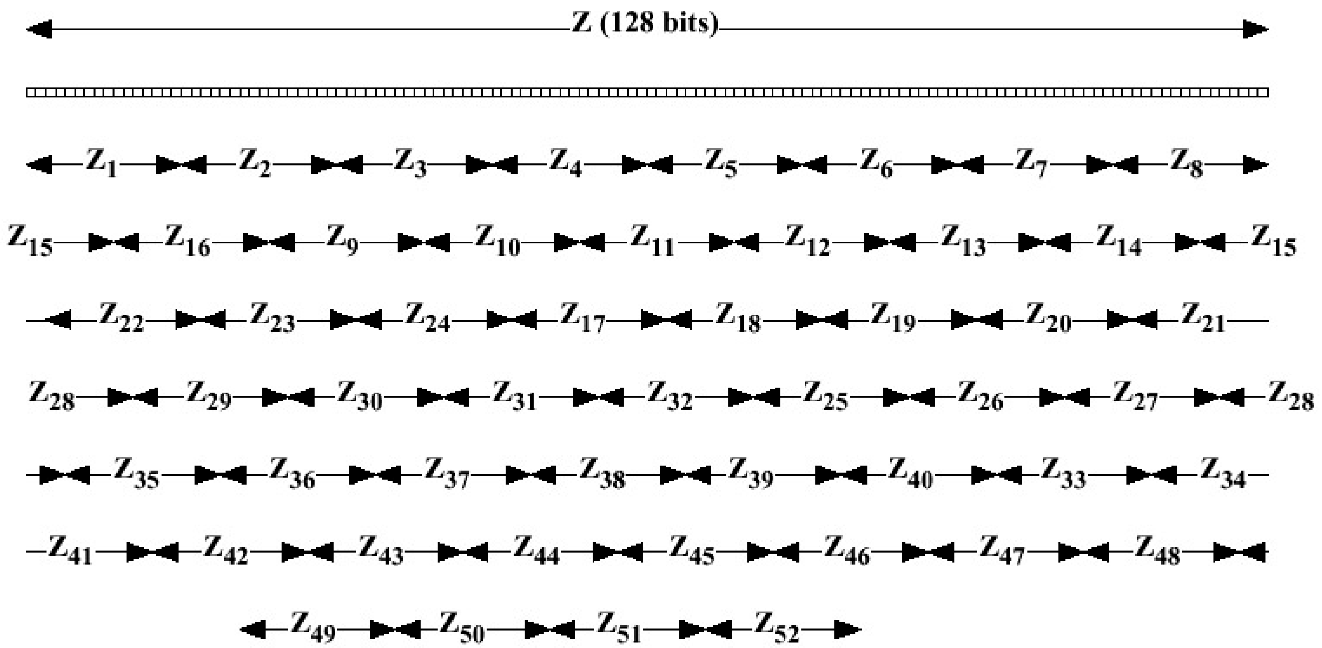
- **Input – 128-bit key Z**
- **Output - 52 16-bit subkeys ($Z_1$, $Z_2$, …, $Z_{52}$)**
- **The first 8 subkeys $Z_1$, $Z_2$, …, $Z_8$ are taken directly from the key Z**
  **$Z_1$ = Z[1..16], $Z_2$=Z[17..32], …, $Z_8$=Z[113..128]**
- **Circular left shift 25 bit positions of Z and extract next 8 subkeys**
- **Repeat the above procedure until all of the 52 subkeys are generated**

# IDEA (cont.)

## IDEA Encryption (cont.)

- Subkey Generation (cont.)

  $Z_1 = Z[1..16]$
  $Z_7 = Z[97..112]$
  $Z_{13} = Z[90..105]$
  $Z_{19} = Z[83..98]$
  $Z_{25} = Z[76..91]$
  $Z_{31} = Z[44..59]$
  $Z_{37} = Z[37..52]$
  $Z_{43} = Z[30..45]$



# IDEA (cont.)

## IDEA Decryption

- Essentially the same as the encryption process
- The decryption keys $U_1$, $U_2$, …, $U_{52}$ are derived from the encryption keys:
  - The first 4 subkeys of decryption round *i* are derived from the first 4 subkeys of round (10 – *i*), where the transformation stage is regarded as round 9:
    - The 1st and 4th decryption subkeys are equal to the multiplicative inverse modulo ($2^{16}$ + 1) of the corresponding 1st and 4th encryption subkeys
    - For rounds 2 through 8, the 2nd and 3rd decryption subkeys are the additive inverse modulo ($2^{16}$) of the 3rd and 2nd encryption subkeys
    - For rounds 1 and 9, the 2nd and 3rd decryption subkeys are the additive inverse modulo ($2^{16}$) of the 2nd and 3rd encryption subkeys
  - For the first 8 rounds, the last 2 subkeys of decryption round *i* are equal to the last 2 subkeys of encryption round (9 – *i*)

# IDEA (cont.)

## IDEA Decryption (cont.)

| | Encryption | | Decryption | |
|---|---|---|---|---|
| Stage | Designation | Equivalent to | Designation | Equivalent to |
| Round 1 | $Z_1\ Z_2\ Z_3\ Z_4\ Z_5\ Z_6$ | Z[1..96] | $U_1\ U_2\ U_3\ U_4\ U_5\ U_6$ | $Z_{49}^{-1}\ -Z_{50}\ -Z_{51}\ Z_{52}^{-1}\ Z_{47}\ Z_{48}$ |
| Round 2 | $Z_7\ Z_8\ Z_9\ Z_{10}\ Z_{11}\ Z_{12}$ | Z[97..128; 26..89] | $U_7\ U_8\ U_9\ U_{10}\ U_{11}\ U_{12}$ | $Z_{43}^{-1}\ -Z_{45}\ -Z_{44}\ Z_{46}^{-1}\ Z_{41}\ Z_{42}$ |
| Round 3 | $Z_{13}\ Z_{14}\ Z_{15}\ Z_{16}\ Z_{17}\ Z_{18}$ | Z[90..128; 1..25; 51..82] | $U_{13}\ U_{14}\ U_{15}\ U_{16}\ U_{17}\ U_{18}$ | $Z_{37}^{-1}\ -Z_{39}\ -Z_{38}\ Z_{40}^{-1}\ Z_{35}\ Z_{36}$ |
| Round 4 | $Z_{19}\ Z_{20}\ Z_{21}\ Z_{22}\ Z_{23}\ Z_{24}$ | Z[83..128;1..50] | $U_{19}\ U_{20}\ U_{21}\ U_{22}\ U_{23}\ U_{24}$ | $Z_{31}^{-1}\ -Z_{33}\ -Z_{32}\ Z_{34}^{-1}\ Z_{29}\ Z_{30}$ |
| Round 5 | $Z_{25}\ Z_{26}\ Z_{27}\ Z_{28}\ Z_{29}\ Z_{30}$ | Z[76..128; 1..43] | $U_{25}\ U_{26}\ U_{27}\ U_{28}\ U_{29}\ U_{30}$ | $Z_{25}^{-1}\ -Z_{27}\ -Z_{26}\ Z_{28}^{-1}\ Z_{23}\ Z_{24}$ |
| Round 6 | $Z_{31}\ Z_{32}\ Z_{33}\ Z_{34}\ Z_{35}\ Z_{36}$ | Z[44..75; 101..128; 1..36] | $U_{31}\ U_{32}\ U_{33}\ U_{34}\ U_{35}\ U_{36}$ | $Z_{19}^{-1}\ -Z_{21}\ -Z_{20}\ Z_{22}^{-1}\ Z_{17}\ Z_{18}$ |
| Round 7 | $Z_{37}\ Z_{38}\ Z_{39}\ Z_{40}\ Z_{41}\ Z_{42}$ | Z[37..100; 126..128; 1..29] | $U_{37}\ U_{38}\ U_{39}\ U_{40}\ U_{41}\ U_{42}$ | $Z_{13}^{-1}\ -Z_{15}\ -Z_{14}\ Z_{16}^{-1}\ Z_{11}\ Z_{12}$ |
| Round 8 | $Z_{43}\ Z_{44}\ Z_{45}\ Z_{46}\ Z_{47}\ Z_{48}$ | Z[30..125] | $U_{43}\ U_{44}\ U_{45}\ U_{46}\ U_{47}\ U_{48}$ | $Z_7^{-1}\ -Z_9\ -Z_8\ Z_{10}^{-1}\ Z_5\ Z_6$ |
| transformation | $Z_{49}\ Z_{50}\ Z_{51}\ Z_{52}$ | Z[23..86] | $U_{49}\ U_{50}\ U_{51}\ U_{52}$ | $Z_1^{-1}\ -Z_2\ -Z_3\ Z_4^{-1}$ |

# Blowfish

- **Design characteristics: 64-bit block of plaintext**
  - **Fast** – encrypts data on 32-bit microprocessors at a rate of 18 clock cycles per byte
  - **Compact** – requires less than 5K of memory
  - **Simple** – easy to implement and determine the strength of the algorithm
  - **Variably secure** – the key length is variable (32 ~ 448 bits)
- **Subkey and S-Box Generation**
  - Key length – 32 ~ 448 bits ( 1 ~ 14 32-bit words), $K_1, K_2, \ldots, K_j$, $1 \le j \le 14$
  - The key is used to generate 18 32-bit subkeys ($P_1, P_2, \ldots, P_{18}$) and 4 8×32 S-boxes:

$$S_{1,0}, S_{1,1}, \ldots, S_{1,255}$$
$$S_{2,0}, S_{2,1}, \ldots, S_{2,255}$$
$$S_{3,0}, S_{3,1}, \ldots, S_{3,255}$$
$$S_{4,0}, S_{4,1}, \ldots, S_{4,255}$$

# Blowfish (cont.)

## Subkey and S-Box Generation (cont.)

1. Initialize the P-array and the 4 S-boxes in order, using the bits of the fractional part of the constant $\pi$
2. Perform a bitwise XOR of the P-array and K-array (assume key length is 14 32-bit words):
$$P_1 = P_1 \oplus K_1,\ P_2 = P_2 \oplus K_2,\ \ldots,\ P_{14} = P_{14} \oplus K_{14},\ \ldots,\ P_{18} = P_{18} \oplus K_{18}$$
3. Encrypt the 64-bit block of all 0s using the current P- and S-arrays; replace $P_1$ and $P_2$ with the output of the encryption:
$$P_1, P_2 = E_{P,S}[0]$$
4. Encrypt the output of step 3 using the current P- and S-arrays; replace $P_3$ and $P_4$ with the output of the encryption:
$$P_3, P_4 = E_{P,S}[P_1 \,\|\, P_2]$$
5. Continue the process to update all elements of P and S

# Blowfish (cont.)

## Encryption and Decryption

- Two primitive operations
  - **Addition**: addition of words is performed modulo $2^{32}$, denoted by +
  - **Bitwise exclusive-OR**: denoted by $\oplus$
- In encryption, $LE_i$ and $RE_i$ are referred to as the left and right half of the data after round *i* has been completed
- Encryption algorithm:

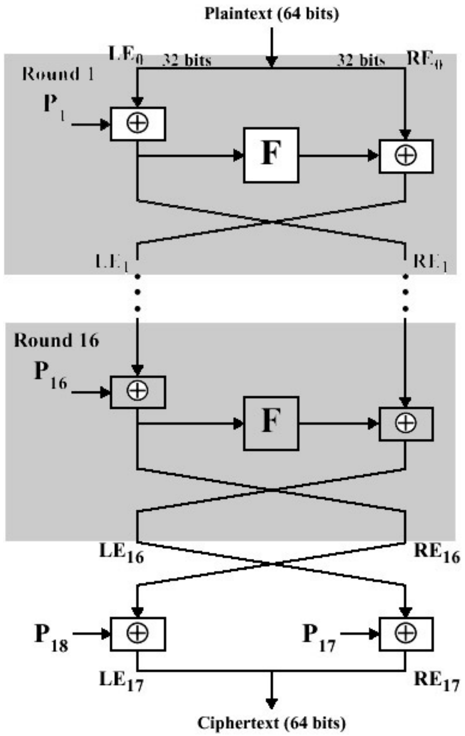
for $i$ = 1 to 16 do

$RE_i = LE_{i-1} \oplus P_i$

$LE_i = F[RE_i] \oplus RE_{i-1}$

$LE_{17} = RE_{16} \oplus P_{18}$

$RE_{17} = LE_{16} \oplus P_{17}$



# Blowfish (cont.)

## Encryption and Decryption (cont.)

- In decryption, $LD_i$ and $RD_i$ are referred to as the left and right half of the data after round
  $i$ has been completed
- Decryption algorithm:

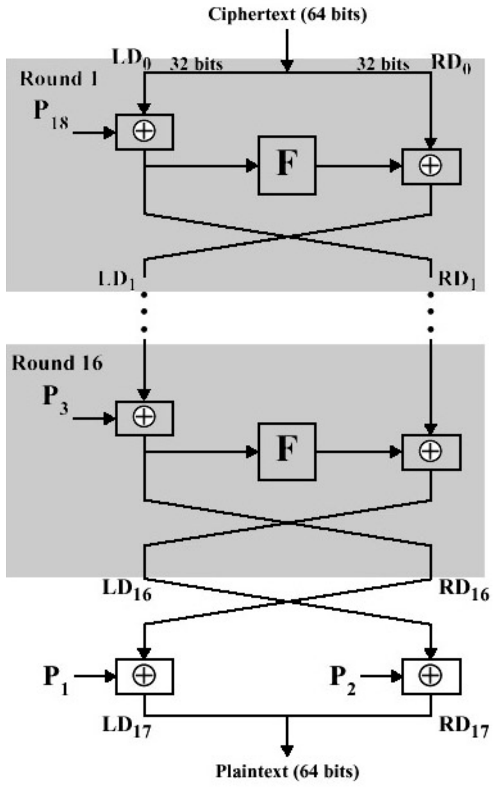
for $i$ = 1 to 16 do

$$RD_i = LD_{i-1} \oplus P_{19-i}$$

$$LD_i = F[RD_i] \oplus RD_{i-1}$$

$$LD_{17} = RD_{16} \oplus P_1$$

$$RD_{17} = LD_{16} \oplus P_2$$

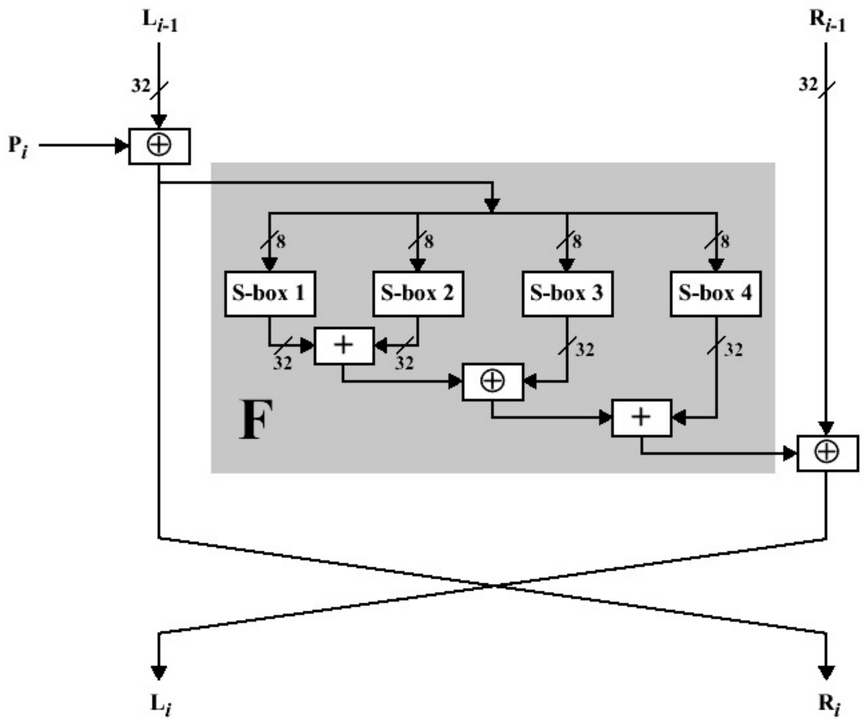


# Blowfish (cont.)

## Encryption and Decryption (cont.)

- Function F: the 32-bit input is divided into 4 bytes (denoted by a, b, c, d)

$$F[a, b, c, d] = ((S_{1,a} + S_{2,b}) \oplus S_{3,c}) + S_{4,d}$$



# Blowfish (cont.)

## Discussion

- The S-boxes and subkeys in blowfish are key dependent
  - Brute-force attack is more difficult
- Operations are performed on both halves of the data in each round
  - Improves the avalanche characteristics of the block cipher
- Fast to execute

**Table 4.3 Speed Comparisons of Block Ciphers on a Pentium**

| Algorithm | Clock cycles per round | # of rounds | # of clock cycles per byte encrypted |
|---|---|---|---|
| Blowfish | 9 | 16 | 18 |
| RC5 | 12 | 16 | 23 |
| DES | 18 | 16 | 45 |
| IDEA | 50 | 8 | 50 |
| Triple-DES | 18 | 48 | 108 |

# RC5

- **Design characteristics:**
  - Suitable for hardware or software – use only primitive operations on microprocessors
  - Fast – the algorithm is word-oriented
  - Adaptable to processors of different word lengths – the number of bits in a word is a parameter of RC5
  - Variable number of rounds – the number of rounds is also a second parameter of RC5, a tradeoff between higher speed and higher security
  - Variable-length key – a third parameter, a tradeoff between speed and security
  - Simple - easy to implement and determine the strength of the algorithm
  - Low memory requirement – suitable for smart cards and other devices with restricted memory
  - High security – provide high security with suitable parameters
  - Data-dependent rotations – strengthen algorithm against cryptanalysis

# RC5 (cont.)

## RC5 Parameters

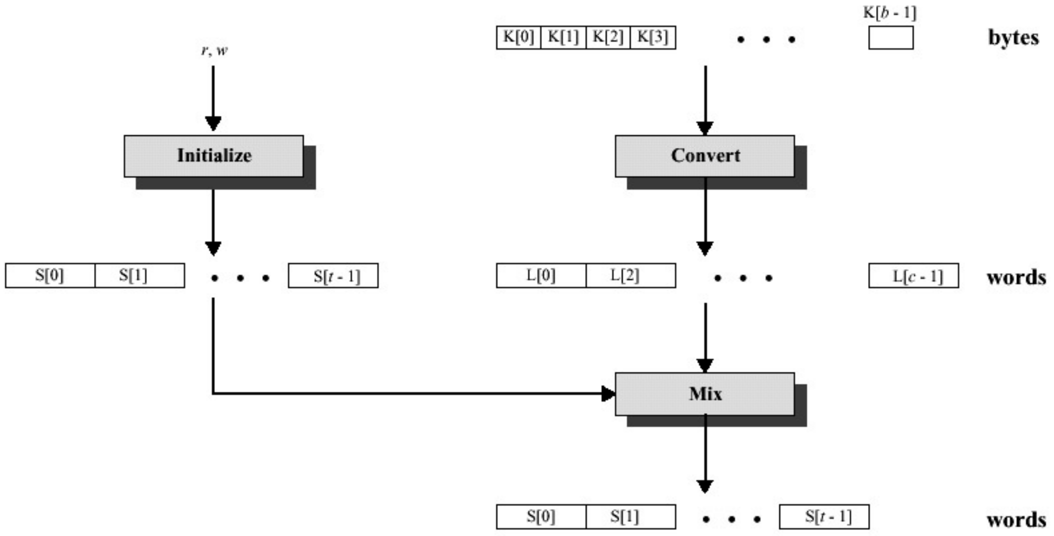
| Parameter | Definition | Allowable Values |
|---|---|---|
| *w* | Word size in bits. RC5 encrypts 2-word blocks | 16, 32, 64 |
| *r* | Number of rounds | 0, 1, …, 255 |
| *b* | Number of 8-bit bytes (octets) in the secret key K | 0, 1, …, 255 |

- Length of the block of plaintext and ciphertext – 32, 64, 128
- Key length – 0 ~ 2040
- A specific version of RC5 is designated as RC5-*w*/*r*/*b*
- Example – RC5-32/12/16 (“nominal” version)
  - 32-bit words (64-bit plaintext and ciphertext blocks)
  - 12 rounds in the encryption and decryption algorithms
  - Key length is 16 bytes (128 bits)

# RC5 (cont.)

## Key Expansion

- A complex set of operations on the secret key to generate *t* subkeys
- Two subkeys are used in each round, and two are on an additional operations, so $t = 2r + 2$
- Each subkey is one word (*w* bits) in length
- Subkeys generation step:
  - Initialization
    S[0], S[1], …, S[*t*-1]
  - Conversion
    K[0 … *b*-1] ⇒ L[0 … *c*-1]
  - Mix



# RC5 (cont.)

## Key Expansion (cont.)

### Initialization

- Define $P_w = \mathrm{Odd}[(e-2)2^w]$
  $Q_w = \mathrm{Odd}[(\phi-1)2^w]$
  where $e = 2.718281828459 \ldots$ (base of natural logarithms)
  $\phi = 1.618033988749 \ldots$ (golden ratio) $= \frac{1+\sqrt{5}}{2}$
  and Odd[$x$] is the odd integer nearest to $x$

| $w$ | 16 | 32 | 64 |
|---|---|---|---|
| $P_w$<br>$Q_w$ | B7E1<br>9E37 | B7E15163<br>9E3779B7 | B7E151628AED2A6B<br>9E3779B77F4A7C15 |

- $S[0] = P_w$
  for $i$ = 1 to $t$-1 do
  $S[i] = S[i-1] + Q_w$

# RC5 (cont.)

## Key Expansion (cont.)

- **Mix operation:**

$i = j = X = Y = 0$

do $3 \times \max(t, c)$ times:

$S[i] = (S[i] + X + Y) \lll 3$

$X = S[i]$

$i = (i + 1) \bmod (t)$

$L[j] = (L[j] + X + Y) \lll (X + Y)$

$Y = L[j]$

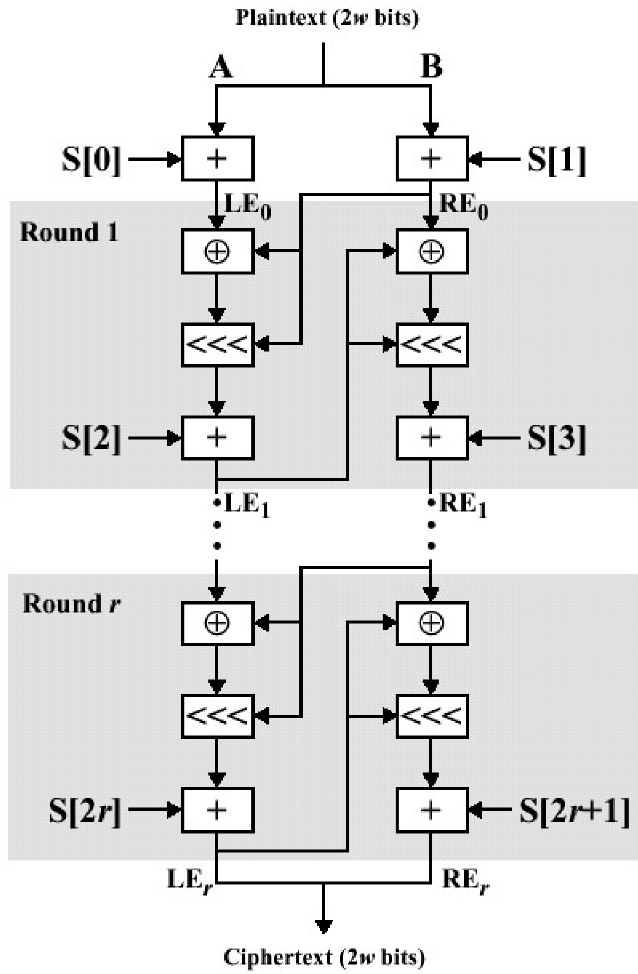
$j = (j + 1) \bmod (c)$

# RC5 (cont.)

## Encryption

- Three primitive operations:
  - Addition
  - Bitwise exclusive-OR
  - Left circular rotation - *x* <<< *y*
- Algorithm:

$LE_0 = A + S[0]$
$RE_0 = B + S[1]$
for $i$ = 1 to $r$ do
$LE_i = ((LE_{i-1} \oplus RE_{i-1}) <<< RE_{i-1}) + S[2i]$
$RE_i = ((RE_{i-1} \oplus LE_i) <<< LE_i) + S[2i + 1]$



# RC5 (cont.)

- **Decryption**
  - Algorithm:

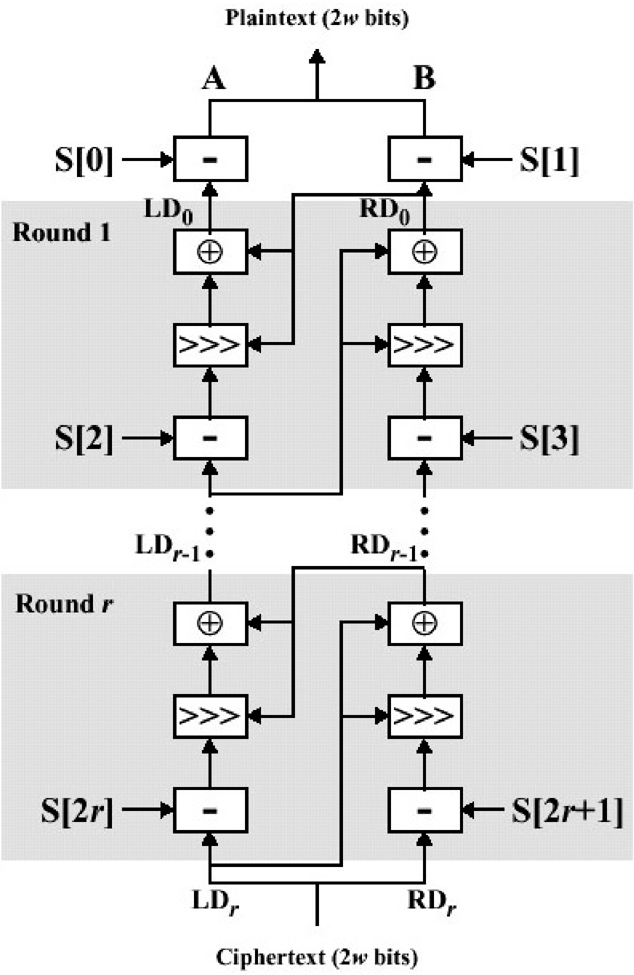
    for $i = r$ down to 1do

    $RD_{i-1} = ((RD_i - S[2i + 1]) >>> LD_i) \oplus LD_i$

    $LD_{i-1} = ((LD_i - S[2i]) >>> RD_{i-1}) \oplus RD_{i-1}$

    $B = RD_0 - S[1]$

    $A = LD_0 - S[0]$



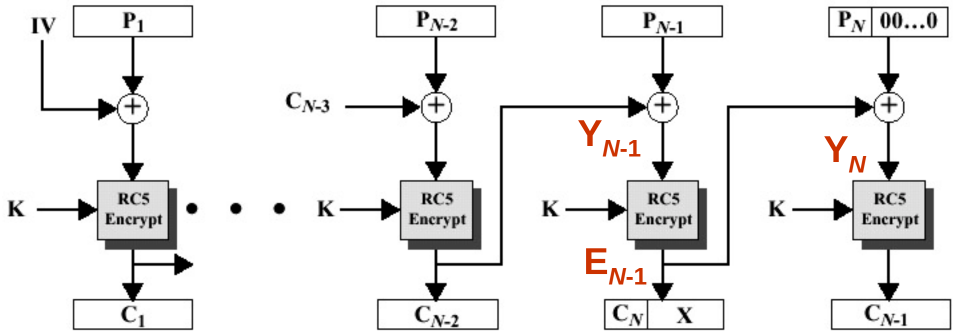
# RC5 (cont.)

- **RC5 Modes (RFC 2040) –**
  - **RC5 block cipher: the electronic codebook (ECB) mode, takes a 2*w*-bit input of plaintext and produces a ciphertext block of size 2*w***
  - **RC5-CBC: the cipher block chaining mode**
  - **RC5-CBC-Pad: a CBC that handles plaintext of any length**
  - **RC5-CTS: the ciphertext stealing mode, handles plaintext of any length and produce ciphertext of equal length**
  - **Padding –**
    - **at the end of message, form 1 to *bb* bytes of padding are added (*bb* = 2*w*/8)**
    - **The pad byte are all the same and are set to a byte that represents the number of bytes of padding, e.g., if there are 8 bytes of padding, each byte has the bit pattern 00001000**

# RC5 (cont.)

## RC5 Modes (cont.)

- Ciphertext stealing Mode – assume the last block is of length $L$, $L < 2w/8$
  1. Encrypt the first ($N$-2) block using the traditional CBC technique
  2. Exclusive-OR $P_{N\text{-}1}$ with $C_{N\text{-}2}$ to create $Y_{N\text{-}1}$
  3. Encrypt $Y_{N\text{-}1}$ to create $E_{N\text{-}1}$
  4. Select the first $L$ bytes of $E_{N\text{-}1}$ to create $C_N$
  5. Pad $P_N$ with 0's at the end and exclusive-OR with $E_{N\text{-}1}$ to create $Y_N$
  6. Encrypt $Y_N$ to create $C_{N\text{-}1}$



# CAST-128

- **Characteristics**
  - Length of the block of plaintext and ciphertext – 64 bits
  - Key length – 40 ~ 128 bits in 8-bit increments
  - 16 rounds of operation
    - Two subkeys in each rounds: 32-bit $Km_i$ and 5-bit $Kr_i$
    - The function F depends on the round
- **Encryption –**
  - Four primitive operations:
    - Addition and subtraction
    - Bitwise exclusive-OR
    - Left circular rotation

# CAST-128 (cont.)

## Encryption (cont.)

- Algorithm:

  $L_0 \,||\, R_0$ = Plaintext

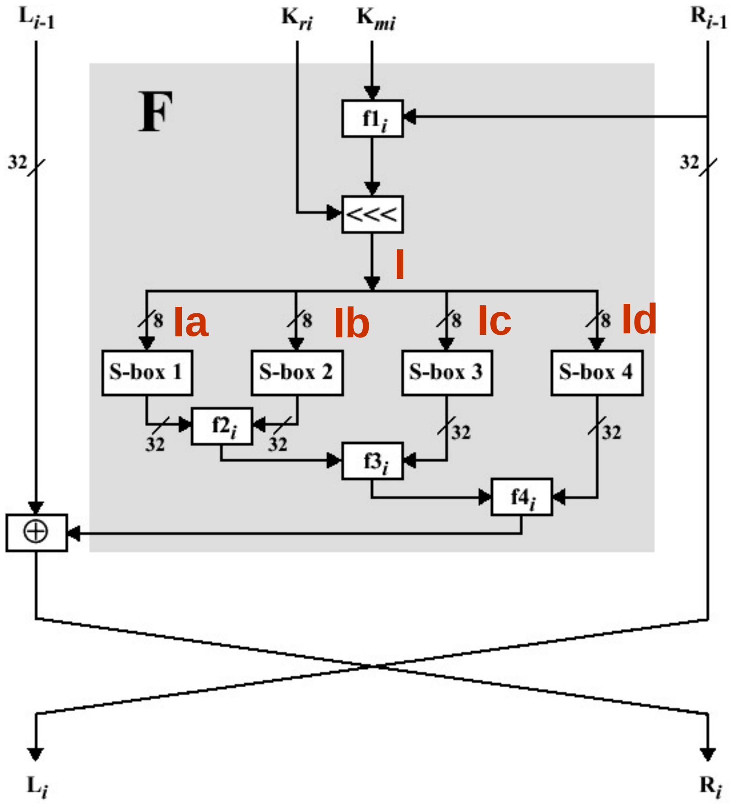
  for $i$ = 1 to 16 do

  $L_i = R_{i-1}$

  $R_i = L_{i-1} \oplus F_i[R_{i-1}, Km_i, Kr_i]$

  Ciphertext = $R_{16} \,||\, L_{16}$

- F function




| | |
|---|---|
| **Rounds 1, 4, 7, 10, 13, 16** | $I = ((Km_i + R_{i-1}) \lll Kr_i)$<br>$F = ((S1[Ia] \oplus S2[Ib]) - S3[Ic]) + S4[Id]$ |
| **Rounds 2, 5, 8, 11, 14** | $I = ((Km_i \oplus R_{i-1}) \lll Kr_i)$<br>$F = ((S1[Ia] - S2[Ib]) + S3[Ic]) \oplus S4[Id]$ |
| **Rounds 3, 6, 9, 12, 15** | $I = ((Km_i - R_{i-1}) \lll Kr_i)$<br>$F = ((S1[Ia] + S2[Ib]) \oplus S3[Ic]) - S4[Id]$ |

# CAST-128 (cont.)

## Encryption (cont.)

### Substitution Boxes

- CAST-128 uses 8 8×32 S-boxes: four (S1 … S4) for encryption and decryption, four (S5 … S8) for subkey generation
- Each S-box is an array of 32 columns by 256 rows
- The 8-bit input select a row and the 32-bit value is the output

### Subkey Generation

- The 128-bit key is labeled : x0x1x2x3x4x5x6x7x8x9xAxBxCxDxEXF
- Define

| | |
|---|---|
| $Km_1, \ldots, Km_{16}$ | 16 32-bit masking subkeys (one per round) |
| $Kr_1, \ldots, Kr_{16}$ | 16 32-bit rotate subkeys, only the least significant 5 bits are used |
| $z_0, \ldots, z_F$ | intermediate (temporary) bytes |
| $K_1, \ldots, K_{32}$ | intermediate (temporary) 32-bit words |

# CAST-128 (cont.)

## Encryption (cont.)

- Subkey Generation (cont.)
  - See Fig. 4.15 for generating K1 … K32 using S5 … S8
  - The subkeys are defined as

    for $i$ = 1 to 16 do

    $Km_i = K_i$

    $Kr_i = K_{16+i}$

# RC2

## RC2 parameters

- Length of the block of plaintext and ciphertext – 64 bits
- Key length – 8 ~ 1024 bits

## Key Expansion

- Generate 128 bytes of subkeys labeled L[0], …, L[127] (16-bit K[0]…K[63])
- Input: *T* bytes of key, put in L[0], …, L[*T*-1]
- Generate pseudorandom bytes P[0], …, P[255] from the digits of $\pi$
- Algorithm:

```
for i = T to 127 do                      /* set L[T] … L[127] */
        L[i] = P[L[i - 1] + L[i - T]]
L[128 – T] = P[L[128 - T]]
for i = 127-T down to 0 do               /* set L[0] … L[127 - T] */
        L[i] = P[L[i + 1] ⊕ L[i + T]]
```

# RC2 (cont.)

## Encryption

- Primitive operations:
  - Addition: +
  - Bitwise exclusive-OR: ⊕
  - Bitwise complement: ~
  - Bitwise AND: &
  - Left circular rotation: $x <<< y$
- Input: 64 bits stored in 16-bit words R[0], R[1], R[2], R[3]
- 18 rounds of mixing and mashing

# RC2 (cont.)

## Encryption (cont.)

- Mixing round:

  R[0] = R[0] + K[*j*] + (R[3] & r[2]) + (~R[3] & R[1])
  R[0] = R[0] <<< 1
  *j* = *j* + 1
  R[1] = R[1] + K[*j*] + (R[0] & r[3]) + (~R[0] & R[2])
  R[1] = R[1] <<< 2
  *j* = *j* + 1
  R[2] = R[2] + K[*j*] + (R[1] & r[0]) + (~R[1] & R[3])
  R[2] = R[2] <<< 3
  *j* = *j* + 1
  R[3] = R[3] + K[*j*] + (R[2] & r[1]) + (~R[2] & R[0])
  R[3] = R[3] <<< 5
  *j* = *j* + 1

K[*j*] is the first subkey that has not yet been used

# RC2 (cont.)

## Encryption (cont.)

- Mashing round:

  R[0] = R[0] + K[R[3] & 63]
  R[1] = R[1] + K[R[0] & 63]
  R[2] = R[2] + K[R[1] & 63]
  R[3] = R[3] + K[R[2] & 63]

- Encryption:

  1. Initialize *j* to 0
  2. Perform 5 mixing rounds (*j* = 20)
  3. Perform 1 mashing round
  4. Perform 6 mixing rounds (*j* = 44)
  5. Perform 1 mashing round
  6. Perform 5 mixing rounds (*j* = 64)

## Characteristics of Advanced Symmetric Block Ciphers

- Variable key length – Blowfish, RC5, CAST-128, RC2
- Mixed operation –
- Data-dependent rotations – RC5
- Key-dependent rotations – CAST-128
- Key-dependent S-boxes – Blowfish
- Lengthy key schedule algorithm – Blowfish
- Variable F – CAST-128
- Variable plaintext/ciphertext block length – RC5
- Variable number of rounds – RC5
- Operations on both data halves each round – IDEA, Blowfish, RC5