Symmetric Ciphers

Why Triple-DES?

- why not Double-DES?
  - NOT same as some other single-DES use, but have
- meet-in-the-middle attack
  - works whenever use a cipher twice
  - since $X = E_{K1}[P] = D_{K2}[C]$
  - attack by encrypting P with all keys and store
  - then decrypt C with keys and match X value
  - can show takes $O(2^{56})$ steps

Triple-DES with Two-Keys

- hence must use 3 encryptions
  - would seem to need 3 distinct keys
- but can use 2 keys with E-D-E sequence
  - $C = E_{K1}[D_{K2}[E_{K1}[P]]]$
  - nb encrypt & decrypt equivalent in security
  - if $K1=K2$ then can work with single DES
- no current known practical attacks

Triple-DES with Three-Keys

- although are no practical attacks on two-key Triple-DES have some indications
- can use Triple-DES with Three-Keys to avoid even these
  - $C = E_{K3}[D_{K2}[E_{K1}[P]]]$
- has been adopted by some Internet applications, eg PGP, S/MIME

Blowfish

- a symmetric block cipher designed by Bruce Schneier in 1993/94
- characteristics
  - fast implementation on 32-bit CPUs
  - compact in use of memory
  - simple structure for analysis/implementation
  - variable security by varying key size
- has been implemented in various products

Blowfish Key Schedule

- uses a 32 to 448 bit key, 32-bit words stored in K-array K_j ,j from 1 to 14
- used to generate
  - 18 32-bit subkeys stored in P array, $P_1 \ldots P_{18}$
  - four 8x32 S-boxes stored in $S_{i,j}$, each with 256 32-bit entries
- Subkeys and S-Boxes Generation:

  1- initialize P-array and then 4 S-boxes **in order** using the fractional part of pi P_1 (left most 32-bit), and so on,,, $S_{4,255.}$

  2- XOR P-array with key-Array (32-bit blocks) and reuse as needed: assume we have up to k_{10} then P_{10} XOR $K_{10,}$, P_{11} XOR K_1 … P_{18} XOR K_8

Blowfish: SubKey and S-Boxes -cont.

3- Encrypt 64-bit block of zeros, and use the result to update P_1 and $P_{2.}$

4- encrypting output form previous step using current P & S and replace P_3 and P_4. Then encrypting current output and use it to update successive pairs of P.

5- After updating all P's (last :P_{17} P_{18}), start updating S values

using the encrypted output from previous step.

- requires 521 encryptions, hence slow in re-keying
- Not suitable for limited-memory applications.

Blowfish Encryption

- uses two main operations: addition modulo 2^{32} , and XOR
- data is divided into two 32-bit halves L_0 & R_0

for i = 1 to 16 do

$R_i = L_{i-1}$ XOR P_i;

$L_i = F[R_i]$ XOR R_{i-1};

$L_{17} = R_{16}$ XOR P_{18};

$R_{17} = L_{16}$ XOR P_{17};

- where

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

Blowfish Encryption/Decryption

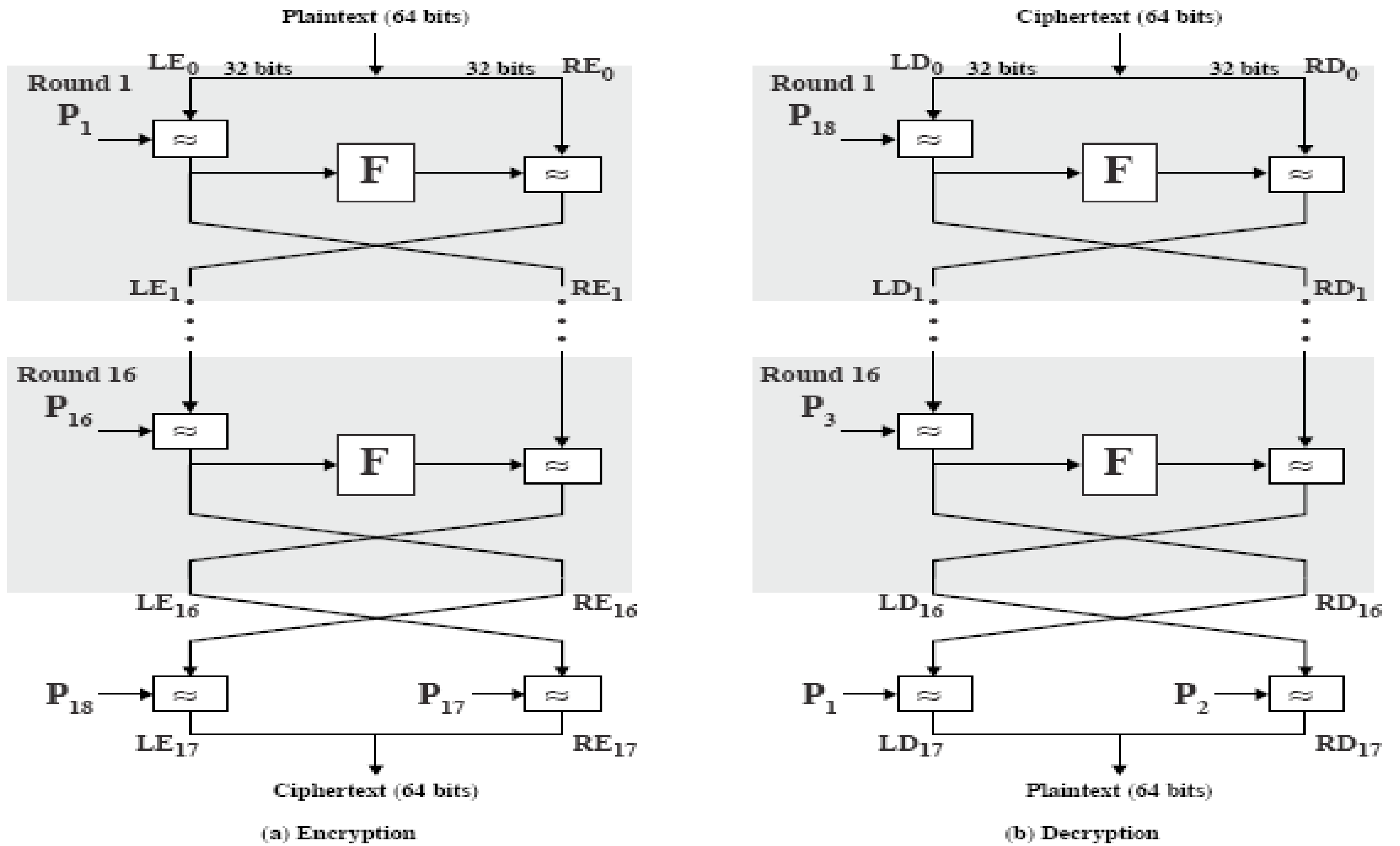


Figure 6.3 Blowfish Encryption and Decryption

Blowfish Encryption

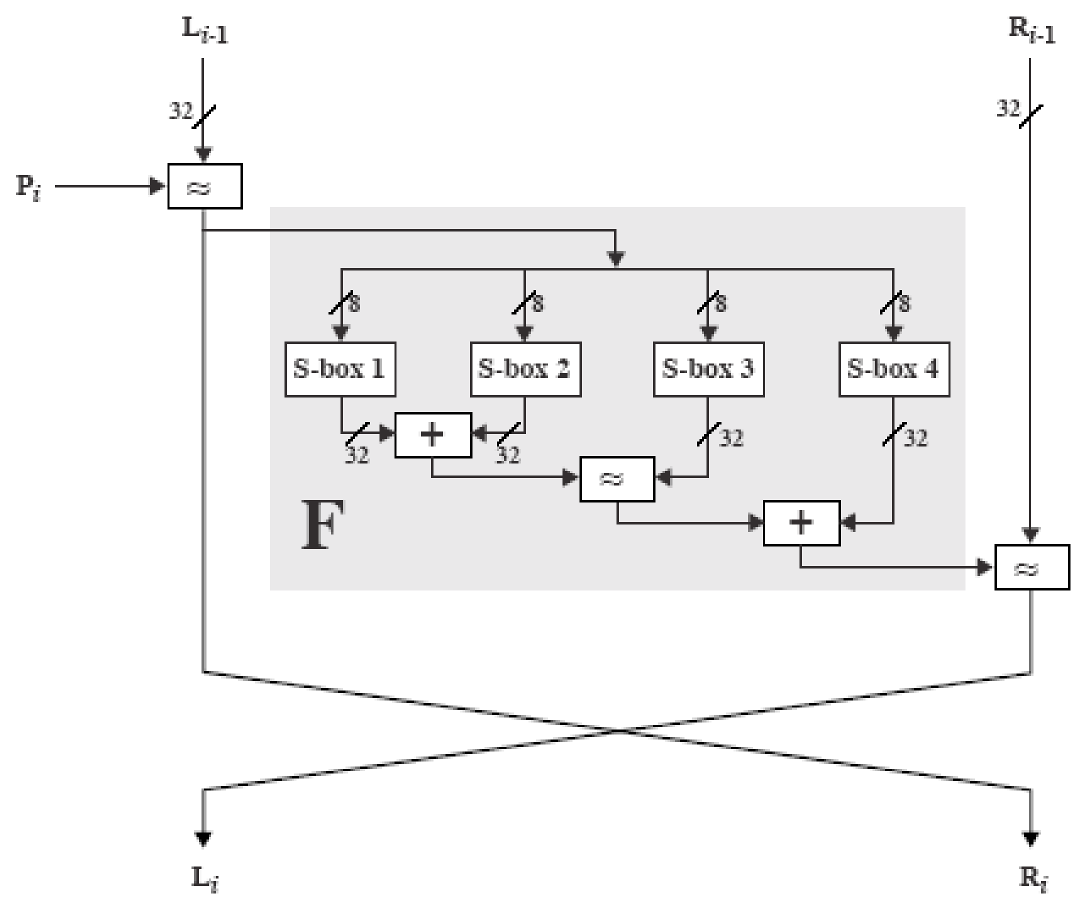


Figure 6.4 Detail of Single Blowfish Round

Discussion

- key dependent S-boxes and subkeys, generated using cipher itself, makes analysis very difficult
- changing both halves in each round increases security
- provided key is large enough, brute-force key search is not practical, especially given the high key schedule cost

RC5

- can vary key size / data size / variable rounds
- very clean and simple design
- easy implementation on various CPUs
- yet still regarded as secure

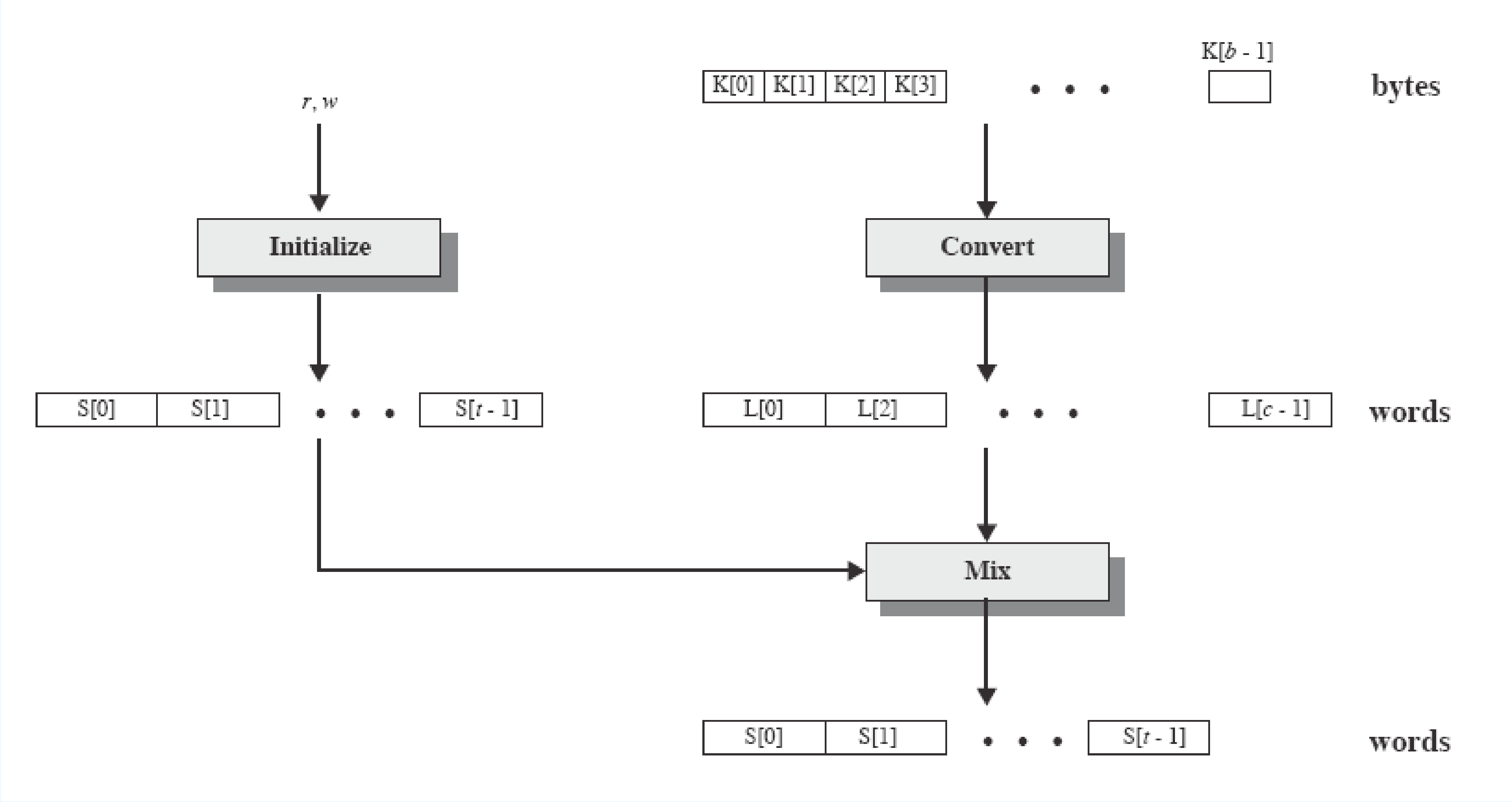
RC5 Ciphers

- RC5 is a family of ciphers RC5-w/r/b
  - w = word size in bits (16/32/64). Encrypts 2w data blocks
  - r = number of rounds (0..255)
  - b = number of bytes in the key (0..255)
- nominal version is RC5-32/12/16
  - ie 32-bit words so encrypts 64-bit data blocks
  - using 12 rounds
  - with 16 bytes (128-bit) secret key

RC5 Key Expansion

- RC5 uses t=2r+2 subkey words (w-bits)
- subkeys are stored in array `S[i]`, i=0..t-1
- then the key schedule consists of
  - initializing S to a fixed pseudorandom value, based on constants e and phi
  - the byte key is copied into a c-words array L
  - a mixing operation then combines L and S to form the final S array

RC5 Key Expansion



RC5 Encryption

- Three main operations: + mod 2^w, XOR, circular left shift <<<, and there inverses used.
- split input into two halves A & B (w-bits each)

  $L_0 = A + S[0];$

  $R_0 = B + S[1];$

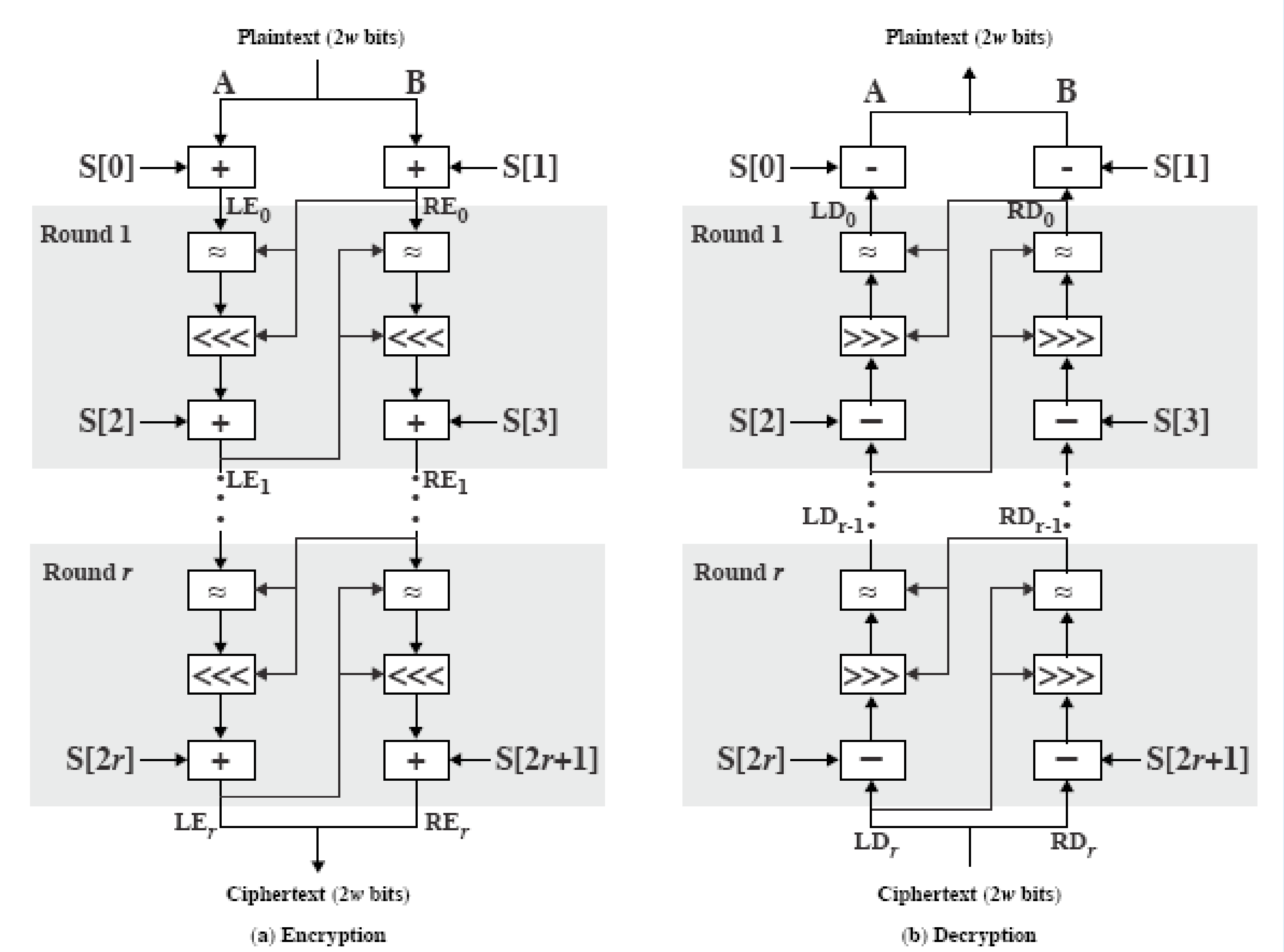
  for i = 1 to r do

  $L_i = ((L_{i-1} \text{ XOR } R_{i-1}) \lll R_{i-1}) + S[2 \times i];$

  $R_i = ((R_{i-1} \text{ XOR } L_i) \lll L_i) + S[2 \times i + 1];$

- each round is like 2 DES rounds
- note rotation is main source of non-linearity
- need reasonable number of rounds (eg 12-16)

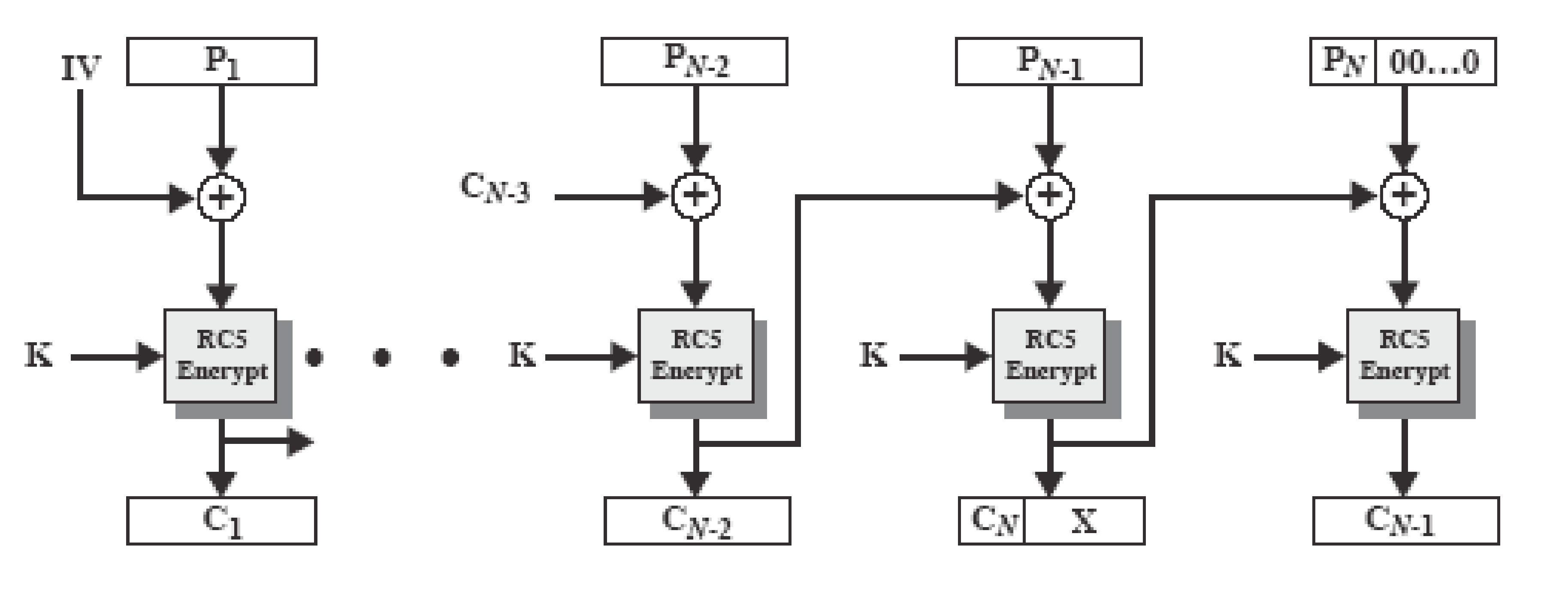
RC5 Encryption



RC5 Modes

- 4 modes used by RC5:
  - RC5 Block Cipher, is ECB mode
  - RC5-CBC, is CBC mode
  - RC5-CBC-PAD, is CBC with padding by bytes with value being the number of padded bytes
  - RC5-CTS, a variant of CBC which is the same size as the original message, uses ciphertext stealing to keep size same as original

RC5 Modes-Ciphertext Stealing (CTS) mode



Block Cipher Characteristics

- features seen in modern block ciphers are:
  - variable key length / block size / rounds
  - mixed operators, data/key dependent rotation
  - key dependent S-boxes
  - more complex key scheduling
  - operation of full data in each round
  - varying non-linear functions

Stream Cipher Properties

- some design considerations are:
  - long period with no repetitions
  - statistically random
  - depends on large enough key
  - confusion
  - diffusion
  - use of highly non-linear boolean functions

RC4

- Designed in 1987 as a proprietary cipher owned by RSA
- simple but effective, widely used: (SSL/TLS standards)
- variable key size (1 to 256 bytes), byte-oriented stream cipher
- key forms random permutation of all 8-bit values
- uses that permutation to scramble input info processed a byte at a time
- fast Software implementations.

RC4 Key Schedule

- starts with an array S of numbers: S[0]=0, …S[255] =255
- Also initialize T with the key. T[i]= K[i mod keylength]
- use key to well and truly shuffle
- S forms **internal state** of the cipher
- given a key k of length l bytes

```
for i = 0 to 255 do
   S[i] = i
j = 0
for i = 0 to 255 do
   j = (j + S[i] + k[i mod l]) (mod 256)
   swap (S[i], S[j])
```

RC4 Encryption

- encryption continues shuffling array values
- sum of shuffled pair selects "stream key" value
- XOR with next byte of message to en/decrypt

```
i = j = 0
for each message byte M_i
    i = (i + 1) (mod 256)
    j = (j + S[i]) (mod 256)
    swap(S[i], S[j])
    t = (S[i] + S[j]) (mod 256)
    C_i = M_i XOR S[t]
```

RC4 Security

- claimed secure against known attacks
    - have some analyses, none practical
- result is very non-linear
- since RC4 is a stream cipher, must **never reuse a key**

Summary

- have considered:
  - some other modern symmetric block ciphers
  - Triple-DES
  - Blowfish
  - RC5
  - briefly introduced stream ciphers
  - RC4