Cryptography and Network Security

Third Edition

by William Stallings

Lecture slides by Lawrie Brown

Chapter 6 – Contemporary Symmetric Ciphers

"I am fairly familiar with all the forms of secret writings, and am myself the author of a trifling monograph upon the subject, in which I analyze one hundred and sixty separate ciphers," said Holmes.

—*The Adventure of the Dancing Men,* Sir Arthur Conan Doyle

Triple DES

- clear a replacement for DES was needed
  - theoretical attacks that can break it
  - demonstrated exhaustive key search attacks
- AES is a new cipher alternative
- prior to this alternative was to use multiple encryption with DES implementations
- Triple-DES is the chosen form

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
- standardized in ANSI X9.17 & ISO8732
- 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 eases analysis/implemention
  - variable security by varying key size
- has been implemented in various products

- Fast: encrypts data on 32 bit MP at a rate of 18 clock cycles per byte.
- Compact: can run in less than 5K of Memory
- Simple: easy to implement.
- Variably secure: key length is variable and can be long as 448 bits.

Blowfish Key Schedule

- uses a 32 to 448 bit key [1 to 14, 32 bit word]
- keys stored in K-array: $K_1, K_2, \ldots K_{14}$
- used to generate
  - 18 32-bit subkeys stored in P-array $P_1, P_2, \ldots P_{18}$
  - four S-boxes each with 256 32-bit entries in $S_{i,j}$

$S_{1,0}; S_{1,1}; \ldots S_{1,255}$

$S_{2,0}; S_{2,1}; \ldots S_{2,255}$

$S_{3,0}; S_{3,1}; \ldots S_{3,255}$

$S_{4,0}; S_{4,1}; \ldots S_{4,255}$

Contd…

- key schedule consists of:
  - initialize P-array and then 4 S-boxes using pi fractional part
    - P1 = 243F6A88
    - P2 = 85A308D3
    - ….
    - S4,254 = 578FDFE3
    - S4,255 = 3AC372E6
  - XOR P-array with key bits (reuse as needed)
  - P1 = P1⊕K1, P2 = P2⊕K2, P3 = P3⊕K3
  - P14 = P14 ⊕K14, P15 = P15 Xor K1, … ,P18 = P18 Xor K4

Contd…

- loop repeatedly encrypting data using current P & S and replace successive pairs of P then S values
- requires 521 encryptions, hence slow in rekeying
- P1, P2 = Ep,s[0];
- P3, P4 = Ep,s[P1||P2];
- P5, P6 = Ep,s[P3||P4];
- ….
- P17,P18= Ep,s[P16||P17];
- S1,0,S1,1 = Ep,s[P17||P18];
- …
- S4,254,S4,255 = Ep,s[S4,252||S4,253];
- Hence not suitable for application that changes key frequently.

Blowfish Encryption

- uses two primitives: addition (add by modulo 2^{32} & 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};

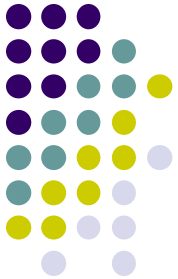
$L_{17} = R_{16}$ XOR P_{18};

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

- where

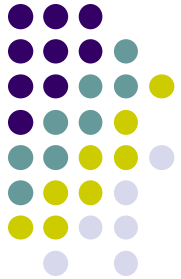
F[a,b,c,d] = (($S_{1,a}$ + $S_{2,b}$) XOR $S_{3,c}$) + $S_{4,d}$

The Blowfish Algorithm



- **There are two parts to this algorithm;**
  - A part that handles the expansion of the key.
  - A part that handles the encryption of the data.
- **The expansion of the key:** break the original key into a set of subkeys. Specifically, a key of no more than 448 bits is separated into 4168 bytes. There is a P-array and four 32-bit S-boxes. The P-array contains 18 32-bit subkeys, while each S-box contains 256 entries.
- **The encryption of the data:** 64-bit input is denoted with an x, while the P-array is denoted with a Pi (where i is the iteration).

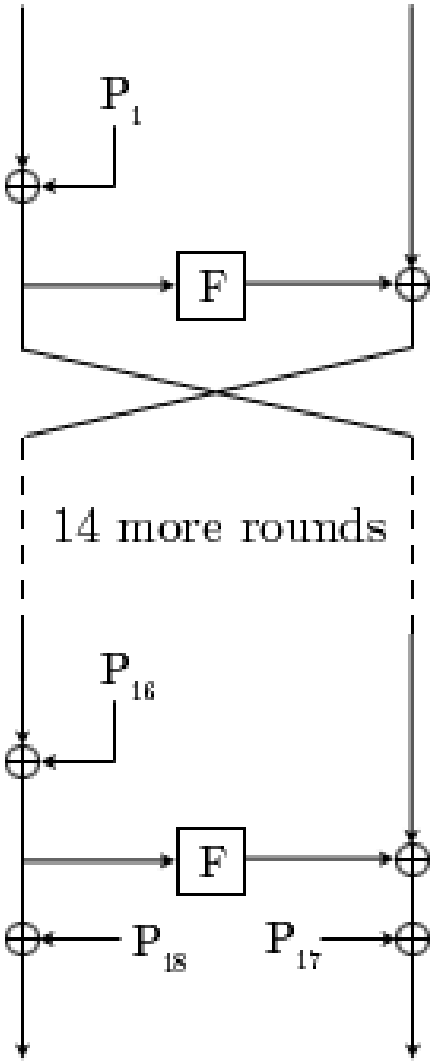
The Blowfish Algorithm: Key Expansion (cont)



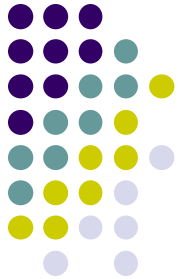
- Blowfish has a 64-bit block size and a key length of anywhere from 32 bits to 448 bits (32-448 bits in steps of 8 bits; default 128 bits).
- It is a 16-round Feistel cipher and uses large key-dependent S-boxes. It is similar in structure to CAST-128, which uses fixed S-boxes.

The Blowfish Algorithm: Key Expansion (cont)

- The diagram to shows the action of Blowfish. Each line represents 32 bits. The algorithm keeps two subkey arrays: the 18-entry P-array and four 256-entry S-boxes.
- The S-boxes accept 8-bit input and produce 32-bit output. One entry of the P-array is used every round, and after the final round, each half of the data block is XORed with one of the two remaining unused P-entries.

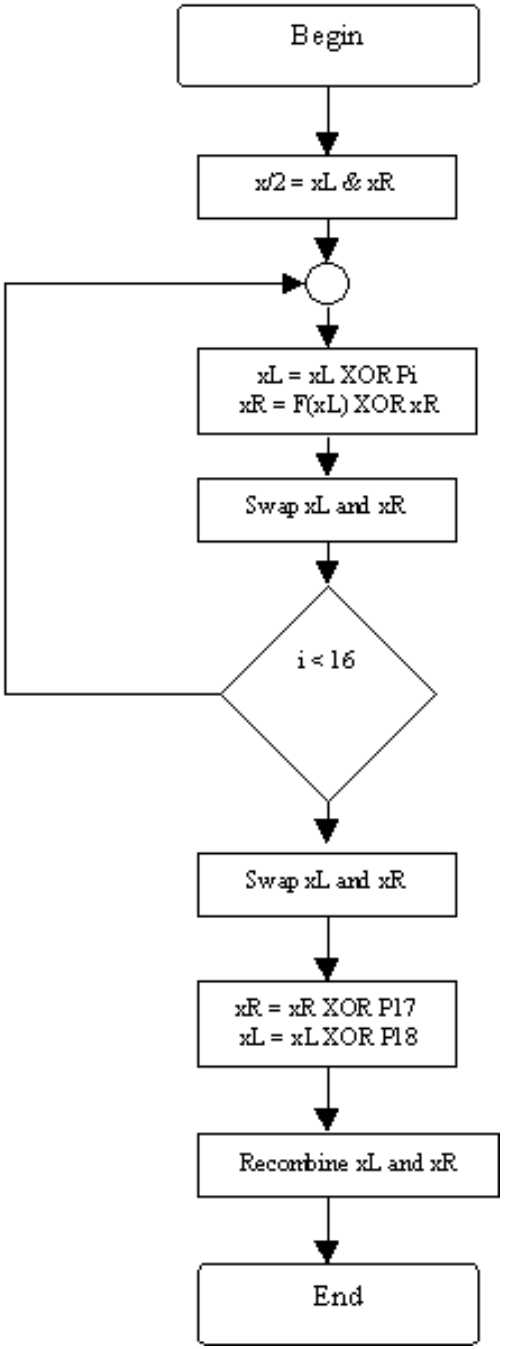


The Blowfish Algorithm: Key Expansion (cont)



- Initialize the P-array and S-boxes
- XOR P-array with the key bits. For example, P1 XOR (first 32 bits of key), P2 XOR (second 32 bits of key), ...
- Use the above method to encrypt the all-zero string
- This new output is now P1 and P2
- Encrypt the new P1 and P2 with the modified subkeys
- This new output is now P3 and P4
- Repeat 521 times in order to calculate new subkeys for the P-array and the four S-boxes

The Blowfish Algorithm



The Blowfish Algorithm: Encryption (cont)

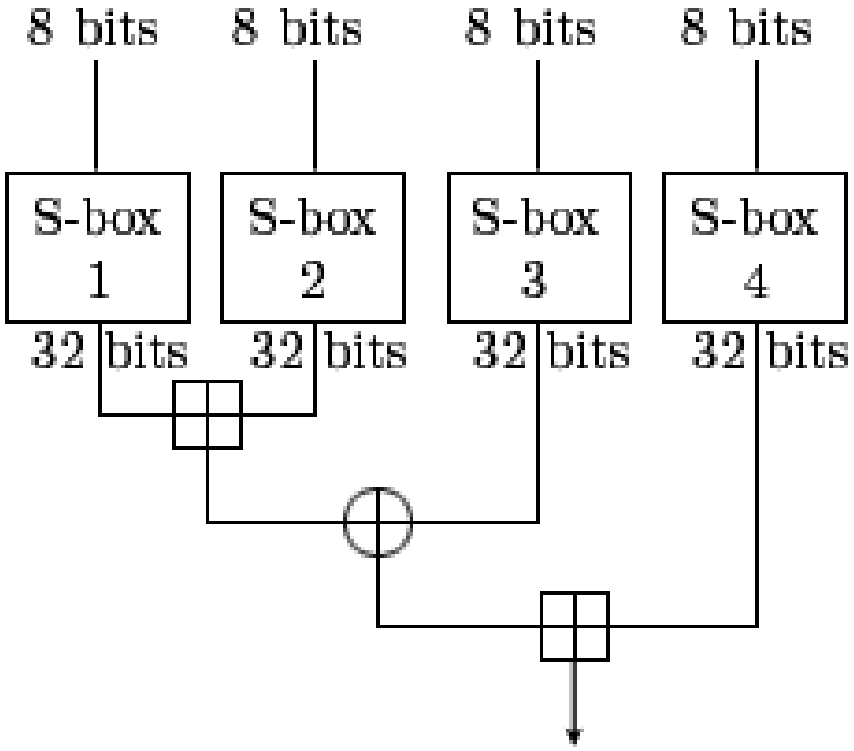
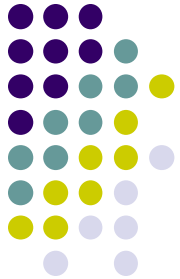
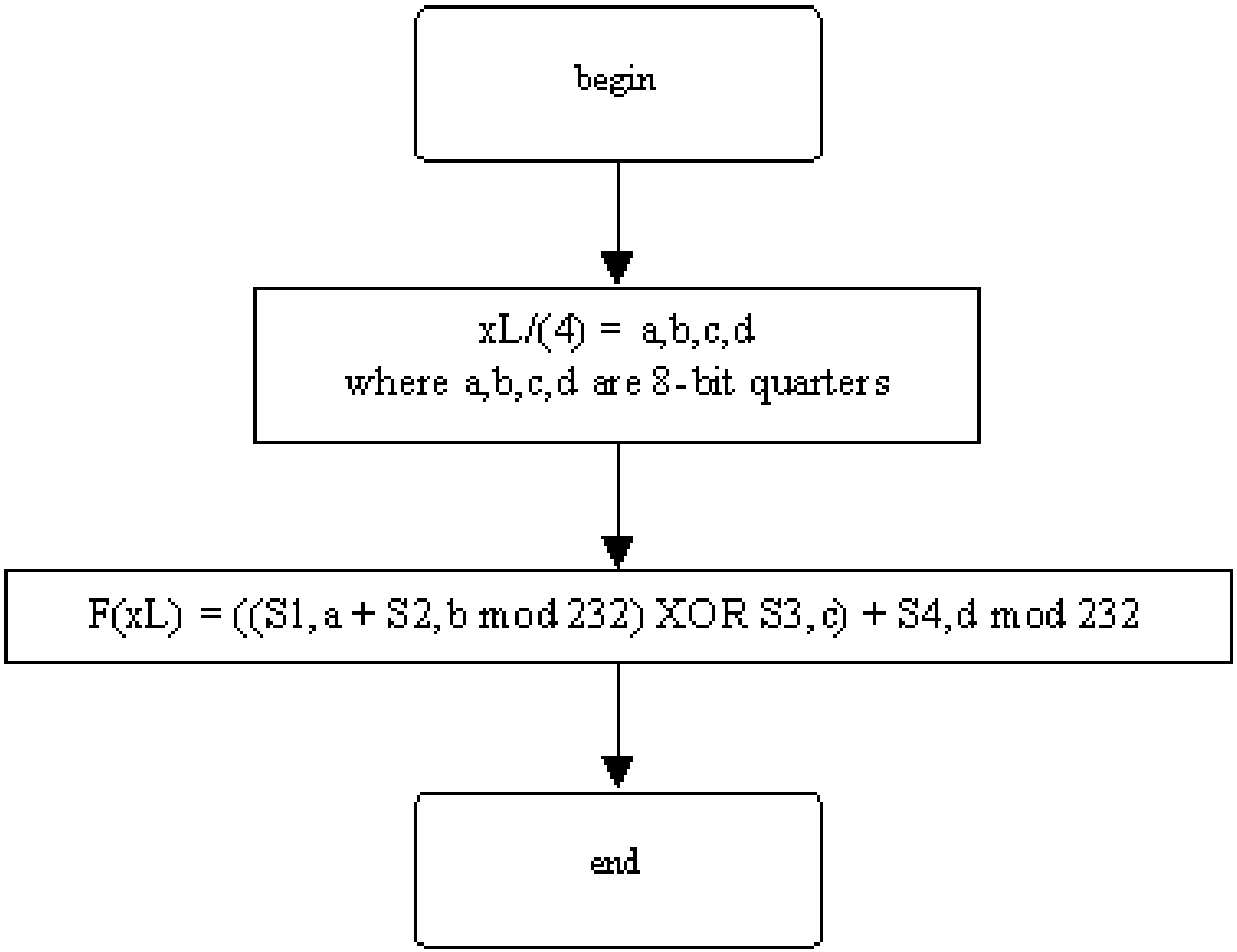


Diagram of Blowfish's F function

The Blowfish Algorithm: Encryption (cont)

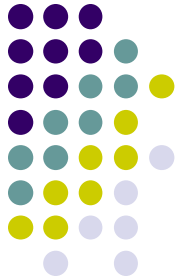


- The diagram to the right shows Blowfish's F-function. The function splits the 32-bit input into four eight-bit quarters, and uses the quarters as input to the S-boxes. The outputs are added modulo 2^{32} and XORed to produce the final 32-bit output.
- Since Blowfish is a Feistel network, it can be inverted simply by XORing P17 and P18 to the ciphertext block, then using the P-entries in reverse order.



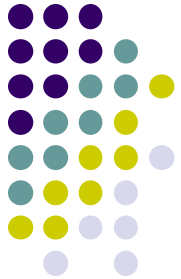
The Function F

The Blowfish Algorithm (cont)



- Blowfish's key schedule starts by initializing the P-array and S-boxes with values derived from the hexadecimal digits of pi, which contain no obvious pattern.
- The secret key is then XORed with the P-entries in order (cycling the key if necessary). A 64-bit all-zero block is then encrypted with the algorithm as it stands.
- The resultant ciphertext replaces P1 and P2. The ciphertext is then encrypted again with the new subkeys, and P3 and P4 are replaced by the new ciphertext. This continues, replacing the entire P-array and all the S-box entries.
- In all, the Blowfish encryption algorithm will run 521 times to generate all the subkeys - about 4KB of data is processed.

DECRYPTION



- Cipher text data is divided into two 32-bit halves

  LD_0 & RD_0

  for i = 1 to 16 do

  $RD_i = LD_{i-1}$ XOR P_{19-i};

  LD_i = F[RD_i] XOR RD_{i-1};

  $LD_{17} = RD_{16}$ XOR P_1;

  $RD_{17} = LD_{16}$ XOR P_2;

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

- a proprietary cipher owned by RSADSI
- designed by Ronald Rivest (of RSA fame)
- used in various RSADSI products
- can vary key size / data size / no 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) nb data=2w
  - r = number of rounds (0..255)
  - b = number of bytes in 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 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 (little-endian) into a c-word array L
  - a mixing operation then combines L and S to form the final S array

RC5 Encryption

- split input into two halves A & B

  $L_0 = A + S[0];$

  $R_0 = B + S[1];$

  for i = 1 to r do

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

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

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

RC5 Modes

- RFC2040 defines 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 padding 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

Block Cipher Characteristics

- features seen in modern block ciphers are:
  - variable key length / block size / no 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 Ciphers

- process the message bit by bit (as a stream)
- typically have a (pseudo) random **stream key**
- combined (XOR) with plaintext bit by bit
- randomness of **stream key** completely destroys any statistically properties in the message
  - $C_i = M_i$ XOR $StreamKey_i$
- what could be simpler!!!!
- but must never reuse stream key
  - otherwise can remove effect and recover messages

Stream Cipher Properties

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

RC4

- a proprietary cipher owned by RSA DSI
- another Ron Rivest design, simple but effective
- variable key size, byte-oriented stream cipher
- widely used (web SSL/TLS, wireless WEP)
- key forms random permutation of all 8-bit values
- uses that permutation to scramble input info processed a byte at a time

RC4 Key Schedule

- starts with an array S of numbers: 0..255
- 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
- tXOR 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**
- have a concern with WEP, but due to key handling rather than RC4 itself

Summary

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