Blowfish

Blowfish is a symmetric block cipher developed by Bruce Schiner. Blowfish was designed to have the following characteristics:

Fast: Blowfish encrypts data on 32-bit microprocessor at rate of 18 clock cycles per byte.

Compact: Blowfish can run in less than 5K of memory.

Simple: Blowfish's simple structure is easy to implement and eases the task of determining the strength of the algorithm.

Variable secure: The key length is variable and can be as long as 448 bits. This allows a tradeoff between higher speed and higher security.

Blowfish encrypts 64-bit blocks of plaintext into 64-bit blocks of ciphertext. Blowfish is implemented in numerous products and has received a fair amount of security.

Subkey and *S*-box generation

Blowfish makes use of a key that ranges from 32 bits to 448 bits (1 to 14 32-bit words). That key is used to generate 18 32-bit subkeys and four 8x32 S-boxes containing a total of 1024 32-bit entries. The total is 1042 32-bit values, or 4168 bytes.

The keys are stored in a *K*-array:

$$K_1, K_2, \ldots, K_j,\ \ 1 \le j \le 14$$

The subkeys are stored in the *P*-array:

$$P_1, P_2, \ldots, P_{18}$$

There are four *S*-boxes each with 256 32-bit entries:

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

The steps in generating *P*-array and *S*-boxes are as follows:

Initialize first the *P*-array and then the four S-boxes in order using the bits of the fractional part of the constant π. Then the leftmost 32 bits of the fractional part of π become P_1, and so on. Note that in binary π has the following view

```
11.
(00100100001111110110101010001000) (10000101101000110000100011010011)
(00010011000110011000101000101110) (00000011011100000111001101000100)
(10100100000010010011100000100010) (00101001100111110011000111010000)
(00001000001011101111101010011000) (11101100010011100110110010001001)
(01000101001010000010000111100110) (00111000110100000001001101110111)
(10111110010101000110011011001111) (00110100111010010000110001101100)
(11000000101011000010100110110111) (11001001011111000101000011011101)
(00111111100001001101010110110101) (10110101010001110000100100010111)
(10010010000101101101010111011001) (10001001011110011111101100011011)

(11010001001100010000101110100110) (10011000110111111011010110101100)
(00101111111111010111001011011011) (11010000000110101101111110110111)
(10111000111000011010111111101101) (01101010001001100111111010010110)
(10111010011111001001000001000101) (11110001001011000111111110011001)
(00100100101000011001100101000111) (10110011100100010110110011110111)
(00001000000000011111001011100010) (10000101100011101111110000010110)
(01100011011010010010000011011000) (01110001010101110100111001101001)
(10100100010110001111111010100011) (11110100100100110011110101111110)
(00001101100101010111010010001111) (01110010100011101011011001011000)
(01110001100010111100110101011000) (10000010000101010100101011101110)
…
```

Thus, P_1 = 00100100001111110110101010001000
P_2 = 10000101101000110000100011010011
…
$S_{1,\ 0}$ = 11010001001100010000101110100110
$S_{1,\ 1}$ = 10011000110111111011010110101100
…

Perform a bitwise XOR of the P-array and the *K*-array, reusing words from the *K*-array as needed. For example, for the maximum length key (14 32-bit words), $P_1 = P_1 \oplus K_1$, $P_2 = P_2 \oplus K_2$, …, $P_{14} = P_{14} \oplus K_{14}$, $P_{15} = P_{15} \oplus K_1$, … , $P_{18} = P_{18} \oplus K_4$.

Encrypt the 64-bit block of all zeros using the current *P*- and *S*-arrays; replace P_1 and P_2 with the output of the encryption.
Encrypt the output of step 3 using the current *P*- and *S*-arrays and replace P_3 and P_4 with the resulting ciphertext.

Continue this procedure to update all elements of *P* and then, in order, all elements of *S*, using at each step the output of the continuously changing Blowfish algorithm.

The update process can be summarized as follows:

P_1 , P_2 = $E_{P,\ S}$ [0]
P_3 , P_4 = $E_{P,\ S}$ [P_1 || P_2]
…
P_{17} , P_{18} = $E_{P,\ S}$ [P_{15} || P_{16}]
$S_{1,\ 0}$, $S_{1,\ 1}$ = $E_{P,\ S}$ [P_{17} || P_{18}]
…
$S_{4,\ 254}$, $S_{4,255}$ = $E_{P,\ S}$ [$S_{4,\ 252}$ || $S_{4,253}$]

where $E_{P,\ S}$ [Y] is the ciphertext produced by encrypting Y using Blowfish with the arrays S and P.

A total of 521 executions of the Blowfish encryption algorithm are required to produce the final S- and P-arrays. Accordingly, Blowfish is not suitable for applications in which the secret key changes frequently. Further, for rapid execution, the S- and P-arrays can be stored rather than rederived from the key each time the algorithm is
used. This requires over 4 KByte of memory.

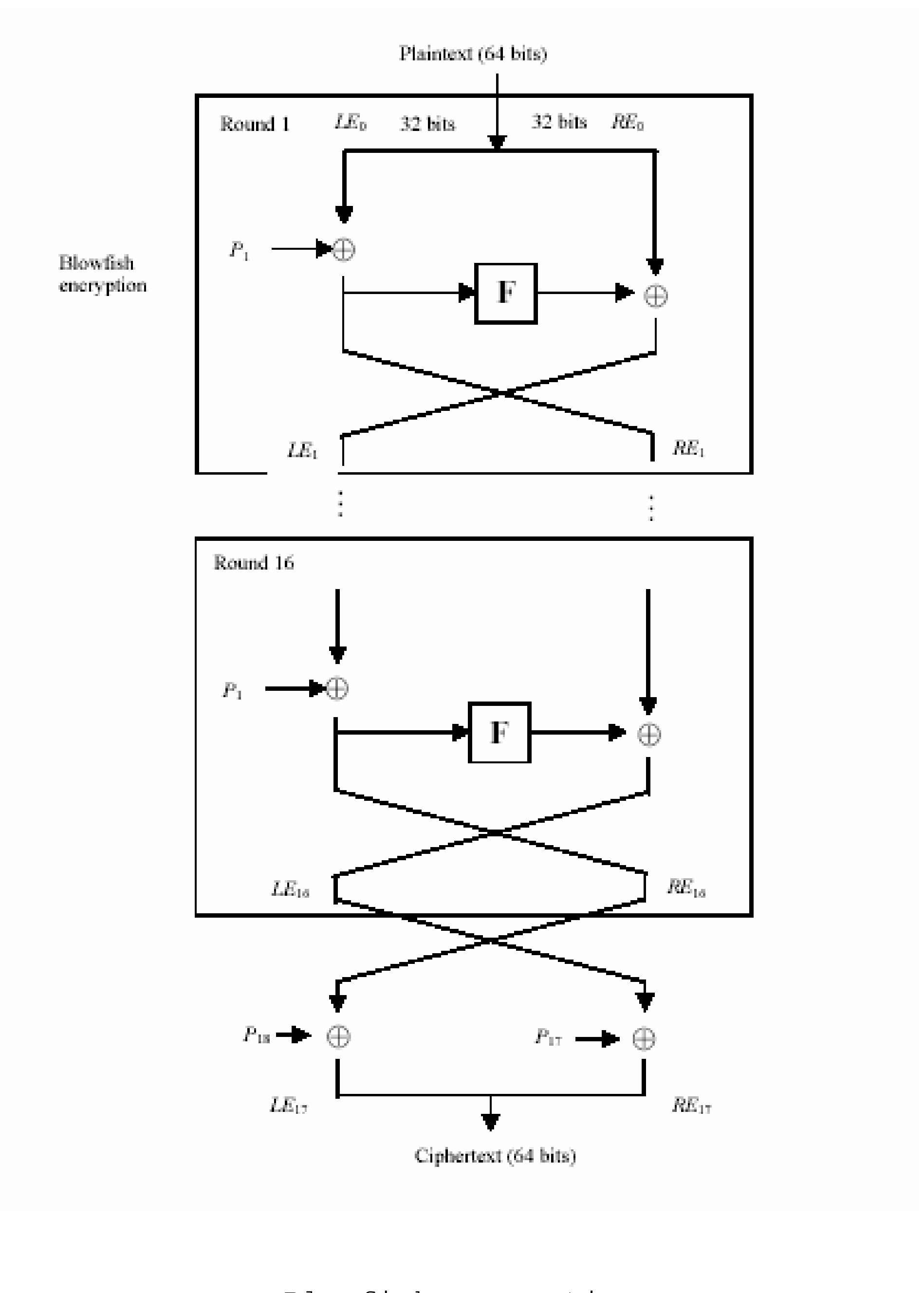
Encryption and decryption

Blowfish uses two primitive operations:

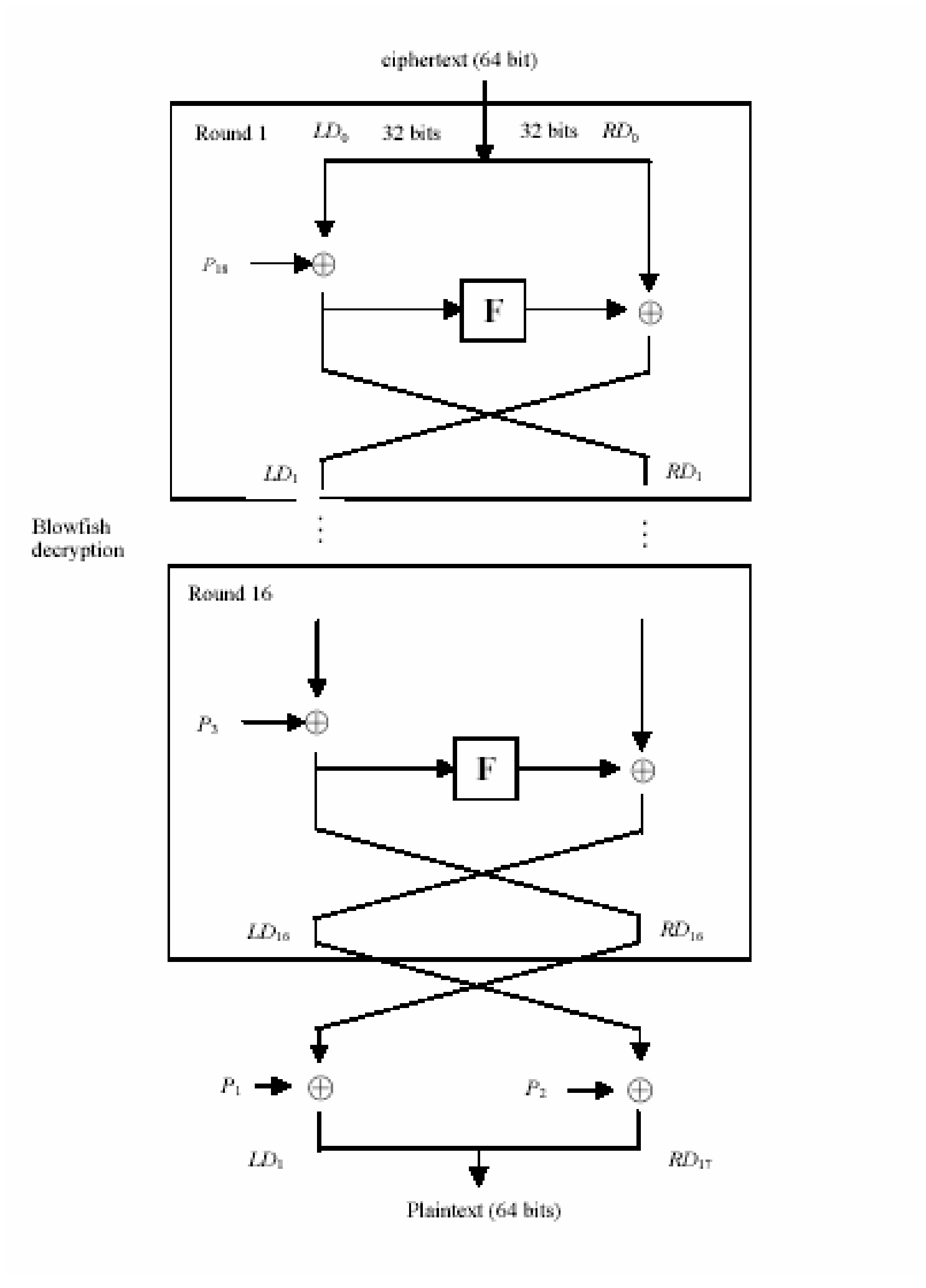
Addition: Addition of words, denoted by +, is performed modulo 2^{32}.
Bitwise exclusive-OR: The operation is denoted by $\oplus$.

It is important that these two operations do not commute. This makes cryptanalysis more difficult.

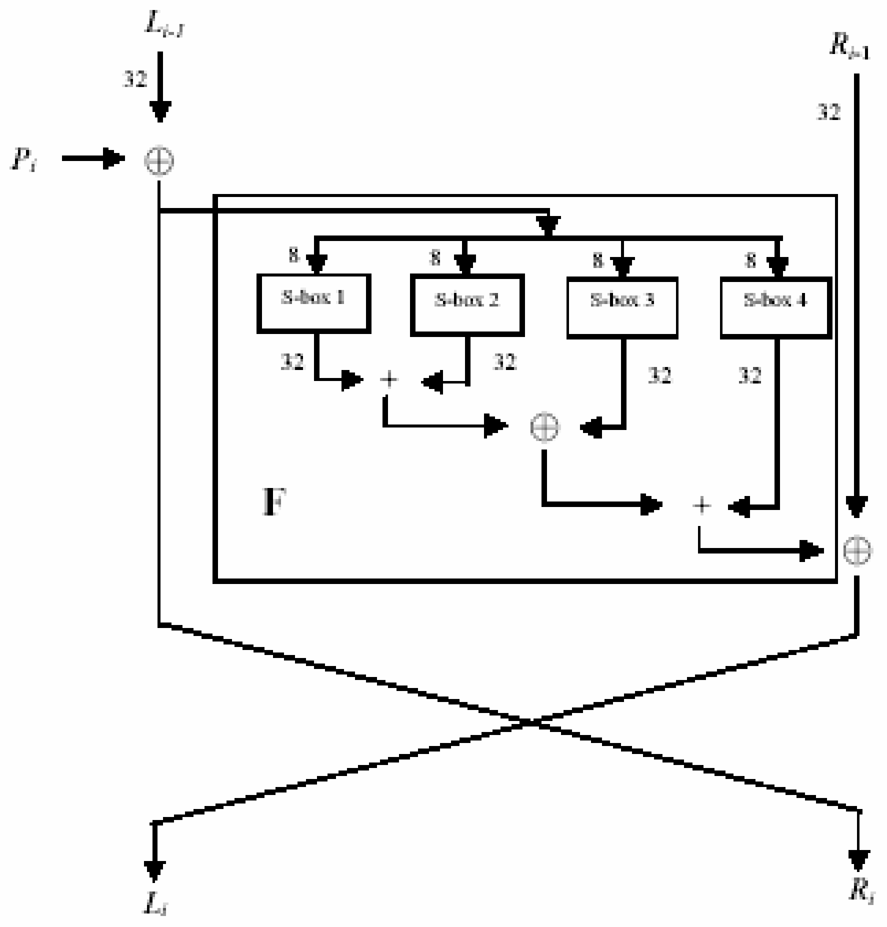
The following two figures are illustrating the encryption and decryption algorithms. The plaintext is divided into two 32-bit halves LE_0 and RE_0. We use the variables LE_i and RE_i to refer to the left and right half of the data after round i has completed. The resulting ciphertext is contained in the two variables LE_{17} and RE_{17}.



Blowfish encryption



Blowfish decryption



Detail of a single Blowfish algorithm

Decryption is easily derived from the encryption algorithm. In this case, the 64-bits of ciphertext ate initially assigned to the two one-word variables LD_0 and RD_0. We use the variables LD_i and RD_i to refer to the left and right half of the data after round *i*. As with most block ciphers, Blowfish decryption involves using the subkeys in reverse order. However, unlike most block ciphers, Blowfish decryption occurs in the same algorithmic direction as encryption, rather than reverse.
The function *F* is shown in the figure below. The 32-bit input to *F* is divided into 4 bytes. If we label those bytes *a*, *b*, *c*, and *d*, then the function can be defined as follows:

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

S-box design

One of the most intense areas of research in the field of symmetric block ciphers is that of S-box design. Here we mention some of the general principles. In essence, we would like any change to the input vector to an S-box to result in random-looking changes to the output. The relationship should be nonlinear and difficult to approximate with linear functions.

One obvious characteristic of the S-box is its size. An *n*x*m* S-box has n input bits and *m* output bits. DES has 6x4 S-boxes. The Blowfish has 8X32 S-boxes. Larger S-boxes, by and large, are more resistant to differential and linear cryptanalysis. On the other hand, the larger the dimension *n*, the larger the lookup table. Thus, for practical reasons, a limit of *n* equal to about 8 to 10 is usually imposed.

S-boxes are typically organized different from the manner used in DES. An *n*x*m* S-box typically consists of 2^n rows and of *m* bits each. The *n* bits of input select one of the rows of the S-box, and the *m* bits in that row are the output. For example, in an 8x32 S-box, if the input is 00001001, the output consists of the 8 bits in row 9 (if the first row is labeled row 0).