Implementing a symmetric cryptographic protocol : DES

The DES protocol allows encryption and decryption of 64-bits blocks, with a 56 bits key. This algorithm uses various tables for substitution, expansion, permutations, and also uses a nonlinear operator : the bitwise XOR operator.

The sequence of DES is the following

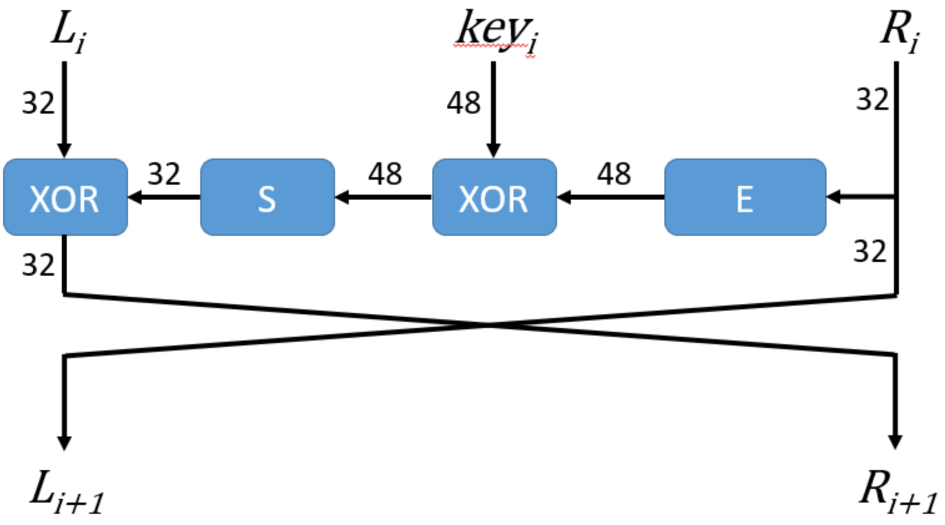
Let *M* be the original message (64 bits block to cipher)

Step 1

M is shuffled using an initial permutation (`init_perm` array)

The resulting block is splitted in two 32-bit blocks: L_0 and R_0

Step 2 is composed of 16 rounds described by the following picture:



2.1 expand R_i using the E table (`expansion_table` array)

2.2 generate 48 bits subkey key_i (see below)

2.3 operate a XOR between expanded version of R_i and key_i

2.4 input the 48 bit computed value at step 2.3 to the ith S-Box (`s_boxes` array), get the 32 bits output

2.5 permute the previous result (`permut_32` array)

2.6 operate a XOR between the result of step 2.5 and L_i

The computed value is then L_{i+1}. Use L_i as initial value for R_{i+1}

Step 3

At the output of the 16th round, the values are L_{16} and R_{16}

L_{16} is appended to R_{16}, then a reverse permutation is done (**`reverse_perm`** array)

Step 2.2 : Generating subkeys for each round.

An arbitrary 56 bits key is chosen, expanded to 64 bits with odd parity control : for each block of 7 bits, an eight bit is added so that the number of 1s in the 8 bit-block is odd.

The key is divided in two blocks of 28 bits each :

The left block *L* is calculated from the key with the **`pc_1_left`** array

The right block *R* is calculated from the key with the **`pc_1_right`** array

Each of these two blocks is then left shifted by some number of positions, depending on the round. For the first round, the shift is 1 bit left, for the second round, the shift is 2 bits left, … See the **`keyshift`** table to know the left shift offset.

The shifted *R* block is then appended to the shifted *L* block, the resulting 48 blocks is finally extracted using the **`pc_2`** array.

This produces the key for the current round.