```
/*****************************************************************************
*                                                                            *
*  -------------------------------- bit.c --------------------------------  *
*                                                                            *
*****************************************************************************/

#include <string.h>

#include "bit.h"

/*****************************************************************************
*                                                                            *
*  ------------------------------ bit_get ------------------------------  *
*                                                                            *
*****************************************************************************/

int bit_get(const unsigned char *bits, int pos) {

unsigned char      mask;

int                i;

/*****************************************************************************
*                                                                            *
*  Set a mask for the bit to get.                                            *
*                                                                            *
*****************************************************************************/

mask = 0x80;

for (i = 0; i < (pos % 8); i++)
   mask = mask >> 1;

/*****************************************************************************
*                                                                            *
*  Get the bit.                                                              *
*                                                                            *
*****************************************************************************/

return (((mask & bits[(int)(pos / 8)]) == mask) ? 1 : 0);

}

/*****************************************************************************
*                                                                            *
*  ------------------------------ bit_set ------------------------------  *
*                                                                            *
*****************************************************************************/

void bit_set(unsigned char *bits, int pos, int state) {

unsigned char      mask;

int                i;

/*****************************************************************************
*                                                                            *
*  Set a mask for the bit to set.                                            *
*                                                                            *
*****************************************************************************/

mask = 0x80;

for (i = 0; i < (pos % 8); i++)
   mask = mask >> 1;

/*****************************************************************************
*                                                                            *
```

```
*  Set the bit.                                                               *
*                                                                             *
******************************************************************************/

if (state)
   bits[pos / 8] = bits[pos / 8] | mask;
else
   bits[pos / 8] = bits[pos / 8] & (~mask);

return;

}

/******************************************************************************
*                                                                             *
*  ------------------------------ bit_xor ------------------------------     *
*                                                                             *
******************************************************************************/

void bit_xor(const unsigned char *bits1, const unsigned char *bits2, unsigned
   char *bitsx, int size) {

int                i;

/******************************************************************************
*                                                                             *
*  Compute the bitwise XOR (exclusive OR) of the two buffers.                 *
*                                                                             *
******************************************************************************/

for (i = 0; i < size; i++) {

   if (bit_get(bits1, i) != bit_get(bits2, i))
      bit_set(bitsx, i, 1);
   else
      bit_set(bitsx, i, 0);

}

return;

}

/******************************************************************************
*                                                                             *
*  ---------------------------- bit_rot_left ----------------------------     *
*                                                                             *
******************************************************************************/

void bit_rot_left(unsigned char *bits, int size, int count) {

int                fbit,
                   lbit,
                   i,
                   j;

/******************************************************************************
*                                                                             *
*  Rotate the buffer to the left the specified number of bits.                *
*                                                                             *
******************************************************************************/

if (size > 0) {

   for (j = 0; j < count; j++) {

      for (i = 0; i <= ((size - 1) / 8); i++) {
```

```
        /*********************************************************************
        *                                                                    *
        *  Get the bit about to be shifted off the current byte.             *
        *                                                                    *
        *********************************************************************/

        lbit = bit_get(&bits[i], 0);

        if (i == 0) {

           /*********************************************************************
           *                                                                    *
           *  Save the bit shifted off the first byte for later.                *
           *                                                                    *
           *********************************************************************/

           fbit = lbit;

           }

        else {

           /*********************************************************************
           *                                                                    *
           *  Set the rightmost bit of the previous byte to the leftmost        *
           *  bit about to be shifted off the current byte.                     *
           *                                                                    *
           *********************************************************************/

           bit_set(&bits[i - 1], 7, lbit);

        }

        /*********************************************************************
        *                                                                    *
        *  Shift the current byte to the left.                               *
        *                                                                    *
        *********************************************************************/

        bits[i] = bits[i] << 1;

      }

      /*********************************************************************
      *                                                                    *
      *  Set the rightmost bit of the buffer to the bit shifted off the    *
      *  first byte.                                                       *
      *                                                                    *
      *********************************************************************/

      bit_set(bits, size - 1, fbit);

   }

}

return;

}
```

```
/*****************************************************************************
*                                                                            *
*  ------------------------------ cbc.c ------------------------------------  *
*                                                                            *
*****************************************************************************/

#include <stdlib.h>

#include "bit.h"
#include "cbc.h"
#include "encrypt.h"

/*****************************************************************************
*                                                                            *
*  ---------------------------- cbc_encipher ---------------------------------  *
*                                                                            *
*****************************************************************************/

void cbc_encipher(const unsigned char *plaintext, unsigned char *ciphertext,
   const unsigned char *key, int size) {

unsigned char      temp[8];

int                i;

/*****************************************************************************
*                                                                            *
*  Encipher the initialization vector.                                       *
*                                                                            *
*****************************************************************************/

des_encipher(&plaintext[0], &ciphertext[0], key);

/*****************************************************************************
*                                                                            *
*  Encipher the buffer using DES in CBC mode.                                *
*                                                                            *
*****************************************************************************/

i = 8;

while (i < size) {

   bit_xor(&plaintext[i], &ciphertext[i - 8], temp, 64);
   des_encipher(temp, &ciphertext[i], NULL);
   i = i + 8;

}

return;

}

/*****************************************************************************
*                                                                            *
*  ---------------------------- cbc_decipher ---------------------------------  *
*                                                                            *
*****************************************************************************/

void cbc_decipher(const unsigned char *ciphertext, unsigned char *plaintext,
   const unsigned char *key, int size) {

unsigned char      temp[8];

int                i;

/*****************************************************************************
*                                                                            *
```

```
*  Decipher the initialization vector.                                    *
*                                                                         *
**************************************************************************/

des_decipher(&ciphertext[0], &plaintext[0], key);

/**************************************************************************
*                                                                         *
*  Decipher the buffer using DES in CBC mode.                             *
*                                                                         *
**************************************************************************/

i = 8;

while (i < size) {

   des_decipher(&ciphertext[i], temp, NULL);
   bit_xor(&ciphertext[i - 8], temp, &plaintext[i], 64);
   i = i + 8;

}

return;

}
```

```
/*****************************************************************************
*                                                                            *
*  ------------------------------ des.c ------------------------------------  *
*                                                                            *
*****************************************************************************/

#include <math.h>
#include <stdlib.h>
#include <string.h>

#include "bit.h"
#include "encrypt.h"

/*****************************************************************************
*                                                                            *
*  Define a mapping for the key transformation.                              *
*                                                                            *
*****************************************************************************/

static const int DesTransform[56] = {

   57, 49, 41, 33, 25, 17,  9,  1, 58, 50, 42, 34, 26, 18,
   10,  2, 59, 51, 43, 35, 27, 19, 11,  3, 60, 52, 44, 36,
   63, 55, 47, 39, 31, 23, 15,  7, 62, 54, 46, 38, 30, 22,
   14,  6, 61, 53, 45, 37, 29, 21, 13,  5, 28, 20, 12,  4

};

/*****************************************************************************
*                                                                            *
*  Define the number of rotations for computing subkeys.                     *
*                                                                            *
*****************************************************************************/

static const int DesRotations[16] = {

   1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1

};

/*****************************************************************************
*                                                                            *
*  Define a mapping for the permuted choice for subkeys.                     *
*                                                                            *
*****************************************************************************/

static const int DesPermuted[48] = {

   14, 17, 11, 24,  1,  5,  3, 28, 15,  6, 21, 10,
   23, 19, 12,  4, 26,  8, 16,  7, 27, 20, 13,  2,
   41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48,
   44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32

};

/*****************************************************************************
*                                                                            *
*  Define a mapping for the initial permutation of data blocks.              *
*                                                                            *
*****************************************************************************/

static const int DesInitial[64] = {

   58, 50, 42, 34, 26, 18, 10,  2, 60, 52, 44, 36, 28, 20, 12,  4,
   62, 54, 46, 38, 30, 22, 14,  6, 64, 56, 48, 40, 32, 24, 16,  8,
   57, 49, 41, 33, 25, 17,  9,  1, 59, 51, 43, 35, 27, 19, 11,  3,
   61, 53, 45, 37, 29, 21, 13,  5, 63, 55, 47, 39, 31, 23, 15,  7
```

```
};

/*****************************************************************************
*                                                                            *
*  Define a mapping for the expansion permutation of data blocks.            *
*                                                                            *
*****************************************************************************/

static const int DesExpansion[48] = {

   32,  1,  2,  3,  4,  5,  4,  5,  6,  7,  8,  9,
    8,  9, 10, 11, 12, 13, 12, 13, 14, 15, 16, 17,
   16, 17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25,
   24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32,  1

};

/*****************************************************************************
*                                                                            *
*  Define tables for the S-box substitutions performed for data blocks.      *
*                                                                            *
*****************************************************************************/

static const int DesSbox[8][4][16] = {

   {
   {14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,  9,  0,  7},
   { 0, 15,  7,  4, 14,  2, 13,  1, 10,  6, 12, 11,  9,  5,  3,  8},
   { 4,  1, 14,  8, 13,  6,  2, 11, 15, 12,  9,  7,  3, 10,  5,  0},
   {15, 12,  8,  2,  4,  9,  1,  7,  5, 11,  3, 14, 10,  0,  6, 13},
   },

   {
   {15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2, 13, 12,  0,  5, 10},
   { 3, 13,  4,  7, 15,  2,  8, 14, 12,  0,  1, 10,  6,  9, 11,  5},
   { 0, 14,  7, 11, 10,  4, 13,  1,  5,  8, 12,  6,  9,  3,  2, 15},
   {13,  8, 10,  1,  3, 15,  4,  2, 11,  6,  7, 12,  0,  5, 14,  9},
   },

   {
   {10,  0,  9, 14,  6,  3, 15,  5,  1, 13, 12,  7, 11,  4,  2,  8},
   {13,  7,  0,  9,  3,  4,  6, 10,  2,  8,  5, 14, 12, 11, 15,  1},
   {13,  6,  4,  9,  8, 15,  3,  0, 11,  1,  2, 12,  5, 10, 14,  7},
   { 1, 10, 13,  0,  6,  9,  8,  7,  4, 15, 14,  3, 11,  5,  2, 12},
   },

   {
   { 7, 13, 14,  3,  0,  6,  9, 10,  1,  2,  8,  5, 11, 12,  4, 15},
   {13,  8, 11,  5,  6, 15,  0,  3,  4,  7,  2, 12,  1, 10, 14,  9},
   {10,  6,  9,  0, 12, 11,  7, 13, 15,  1,  3, 14,  5,  2,  8,  4},
   { 3, 15,  0,  6, 10,  1, 13,  8,  9,  4,  5, 11, 12,  7,  2, 14},
   },

   {
   { 2, 12,  4,  1,  7, 10, 11,  6,  8,  5,  3, 15, 13,  0, 14,  9},
   {14, 11,  2, 12,  4,  7, 13,  1,  5,  0, 15, 10,  3,  9,  8,  6},
   { 4,  2,  1, 11, 10, 13,  7,  8, 15,  9, 12,  5,  6,  3,  0, 14},
   {11,  8, 12,  7,  1, 14,  2, 13,  6, 15,  0,  9, 10,  4,  5,  3},
   },

   {
   {12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,  4, 14,  7,  5, 11},
   {10, 15,  4,  2,  7, 12,  9,  5,  6,  1, 13, 14,  0, 11,  3,  8},
   { 9, 14, 15,  5,  2,  8, 12,  3,  7,  0,  4, 10,  1, 13, 11,  6},
   { 4,  3,  2, 12,  9,  5, 15, 10, 11, 14,  1,  7,  6,  0,  8, 13},
   },

   {
```

```
   { 4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,  7,  5, 10,  6,  1},
   {13,  0, 11,  7,  4,  9,  1, 10, 14,  3,  5, 12,  2, 15,  8,  6},
   { 1,  4, 11, 13, 12,  3,  7, 14, 10, 15,  6,  8,  0,  5,  9,  2},
   { 6, 11, 13,  8,  1,  4, 10,  7,  9,  5,  0, 15, 14,  2,  3, 12},
   },

   {
   {13,  2,  8,  4,  6, 15, 11,  1, 10,  9,  3, 14,  5,  0, 12,  7},
   { 1, 15, 13,  8, 10,  3,  7,  4, 12,  5,  6, 11,  0, 14,  9,  2},
   { 7, 11,  4,  1,  9, 12, 14,  2,  0,  6, 10, 13, 15,  3,  5,  8},
   { 2,  1, 14,  7,  4, 10,  8, 13, 15, 12,  9,  0,  3,  5,  6, 11},
   },

};

/*****************************************************************************
*                                                                            *
*  Define a mapping for the P-box permutation of data blocks.                *
*                                                                            *
*****************************************************************************/

static const int DesPbox[32] = {

   16,  7, 20, 21, 29, 12, 28, 17,  1, 15, 23, 26,  5, 18, 31, 10,
    2,  8, 24, 14, 32, 27,  3,  9, 19, 13, 30,  6, 22, 11,  4, 25

};

/*****************************************************************************
*                                                                            *
*  Define a mapping for the final permutation of data blocks.                *
*                                                                            *
*****************************************************************************/

static const int DesFinal[64] = {

   40,  8, 48, 16, 56, 24, 64, 32, 39,  7, 47, 15, 55, 23, 63, 31,
   38,  6, 46, 14, 54, 22, 62, 30, 37,  5, 45, 13, 53, 21, 61, 29,
   36,  4, 44, 12, 52, 20, 60, 28, 35,  3, 43, 11, 51, 19, 59, 27,
   34,  2, 42, 10, 50, 18, 58, 26, 33,  1, 41,  9, 49, 17, 57, 25

};

/*****************************************************************************
*                                                                            *
*  Define a type for whether to encipher or decipher data.                   *
*                                                                            *
*****************************************************************************/

typedef enum DesEorD_ {encipher, decipher} DesEorD;

/*****************************************************************************
*                                                                            *
*  ------------------------------ permute ------------------------------    *
*                                                                            *
*****************************************************************************/

static void permute(unsigned char *bits, const int *mapping, int n) {

   unsigned char      temp[8];

   int                i;

   /**************************************************************************
   *                                                                         *
   *  Permute the buffer using an n-entry mapping.                           *
   *                                                                         *
   **************************************************************************/
```

```
   memset(temp, 0, (int)ceil(n / 8));

   for (i = 0; i < n; i++)
      bit_set(temp, i, bit_get(bits, mapping[i] - 1));

   memcpy(bits, temp, (int)ceil(n / 8));

   return;

}

/*****************************************************************************
*                                                                            *
*  ------------------------------ des_main ------------------------------   *
*                                                                            *
*****************************************************************************/

static int des_main(const unsigned char *source, unsigned char *target, const
   unsigned char *key, DesEorD direction) {

   static unsigned char subkeys[16][7];

   unsigned char        temp[8],
                        lkey[4],
                        rkey[4],
                        lblk[6],
                        rblk[6],
                        fblk[6],
                        xblk[6],
                        sblk;

   int                  row,
                        col,
                        i,
                        j,
                        k,
                        p;

   /*************************************************************************
   *                                                                        *
   *  If key is NULL, use the subkeys as computed in a previous call.       *
   *                                                                        *
   *************************************************************************/

   if (key != NULL) {

      /**********************************************************************
      *                                                                     *
      *  Make a local copy of the key.                                      *
      *                                                                     *
      **********************************************************************/

      memcpy(temp, key, 8);

      /**********************************************************************
      *                                                                     *
      *  Permute and compress the key into 56 bits.                         *
      *                                                                     *
      **********************************************************************/

      permute(temp, DesTransform, 56);

      /**********************************************************************
      *                                                                     *
      *  Split the key into two 28-bit blocks.                              *
      *                                                                     *
      **********************************************************************/
```

```
      memset(lkey, 0, 4);
      memset(rkey, 0, 4);

      for (j = 0; j < 28; j++)
         bit_set(lkey, j, bit_get(temp, j));

      for (j = 0; j < 28; j++)
         bit_set(rkey, j, bit_get(temp, j + 28));

      /*************************************************************************
      *                                                                        *
      *  Compute the subkeys for each round.                                   *
      *                                                                        *
      *************************************************************************/

      for (i = 0; i < 16; i++) {

         /*********************************************************************
         *                                                                    *
         *  Rotate each block according to its round.                         *
         *                                                                    *
         *********************************************************************/

         bit_rot_left(lkey, 28, DesRotations[i]);
         bit_rot_left(rkey, 28, DesRotations[i]);

         /*********************************************************************
         *                                                                    *
         *  Concatenate the blocks into a single subkey.                      *
         *                                                                    *
         *********************************************************************/

         for (j = 0; j < 28; j++)
            bit_set(subkeys[i], j, bit_get(lkey, j));

         for (j = 0; j < 28; j++)
            bit_set(subkeys[i], j + 28, bit_get(rkey, j));

         /*********************************************************************
         *                                                                    *
         *  Do the permuted choice permutation.                               *
         *                                                                    *
         *********************************************************************/

         permute(subkeys[i], DesPermuted, 48);

      }

   }

   /*************************************************************************
   *                                                                        *
   *  Make a local copy of the source text.                                 *
   *                                                                        *
   *************************************************************************/

   memcpy(temp, source, 8);

   /*************************************************************************
   *                                                                        *
   *  Do the initial permutation.                                           *
   *                                                                        *
   *************************************************************************/

   permute(temp, DesInitial, 64);

   /*************************************************************************
```

```
    *                                                                      *
    *  Split the source text into a left and right block of 32 bits.       *
    *                                                                      *
    ************************************************************************/

   memcpy(lblk, &temp[0], 4);
   memcpy(rblk, &temp[4], 4);

   /************************************************************************
    *                                                                      *
    *  Encipher or decipher the source text.                               *
    *                                                                      *
    ************************************************************************/

   for (i = 0; i < 16; i++) {

      /*********************************************************************
       *                                                                   *
       *  Begin the computation of f.                                      *
       *                                                                   *
       *********************************************************************/

      memcpy(fblk, rblk, 4);

      /*********************************************************************
       *                                                                   *
       *  Permute and expand the copy of the right block into 48 bits.    *
       *                                                                   *
       *********************************************************************/

      permute(fblk, DesExpansion, 48);

      /*********************************************************************
       *                                                                   *
       *  Apply the appropriate subkey for the round.                      *
       *                                                                   *
       *********************************************************************/

      if (direction == encipher) {

         /******************************************************************
          *                                                                *
          *  For enciphering, subkeys are applied in increasing order.     *
          *                                                                *
          ******************************************************************/

         bit_xor(fblk, subkeys[i], xblk, 48);
         memcpy(fblk, xblk, 6);

         }

      else {

         /******************************************************************
          *                                                                *
          *  For deciphering, subkeys are applied in decreasing order.     *
          *                                                                *
          ******************************************************************/

         bit_xor(fblk, subkeys[15 - i], xblk, 48);
         memcpy(fblk, xblk, 6);

      }

      /*********************************************************************
       *                                                                   *
       *  Do the S-box substitutions.                                      *
       *                                                                   *
```

```
      ***************************************************************************/

      p = 0;

      for (j = 0; j < 8; j++) {

         /*************************************************************************
         *                                                                        *
         *  Compute a row and column into the S-box tables.                       *
         *                                                                        *
         *************************************************************************/

         row = (bit_get(fblk, (j * 6)+0) * 2) + (bit_get(fblk, (j * 6)+5) * 1);
         col = (bit_get(fblk, (j * 6)+1) * 8) + (bit_get(fblk, (j * 6)+2) * 4) +
               (bit_get(fblk, (j * 6)+3) * 2) + (bit_get(fblk, (j * 6)+4) * 1);

         /*************************************************************************
         *                                                                        *
         *  Do the S-box substitution for the current six-bit block.              *
         *                                                                        *
         *************************************************************************/

         sblk = (unsigned char)DesSbox[j][row][col];

         for (k = 4; k < 8; k++) {

            bit_set(fblk, p, bit_get(&sblk, k));
            p++;

         }

      }

      /*************************************************************************
      *                                                                        *
      *  Do the P-box permutation to complete f.                               *
      *                                                                        *
      *************************************************************************/

      permute(fblk, DesPbox, 32);

      /*************************************************************************
      *                                                                        *
      *  Compute the XOR of the left block and f.                              *
      *                                                                        *
      *************************************************************************/

      bit_xor(lblk, fblk, xblk, 32);

      /*************************************************************************
      *                                                                        *
      *  Set the left block for the round.                                     *
      *                                                                        *
      *************************************************************************/

      memcpy(lblk, rblk, 4);

      /*************************************************************************
      *                                                                        *
      *  Set the right block for the round.                                    *
      *                                                                        *
      *************************************************************************/

      memcpy(rblk, xblk, 4);

   }

   /*************************************************************************
```

```
   *                                                                      *
   *  Set the target text to the rejoined final right and left blocks.    *
   *                                                                      *
   ************************************************************************/

   memcpy(&target[0], rblk, 4);
   memcpy(&target[4], lblk, 4);

   /************************************************************************
   *                                                                      *
   *  Do the final permutation.                                           *
   *                                                                      *
   ************************************************************************/

   permute(target, DesFinal, 64);

   return 0;

}

/************************************************************************
*                                                                       *
*  ---------------------------- des_encipher ----------------------------  *
*                                                                       *
************************************************************************/

void des_encipher(const unsigned char *plaintext, unsigned char *ciphertext,
   const unsigned char *key) {

   des_main(plaintext, ciphertext, key, encipher);

   return;

}

/************************************************************************
*                                                                       *
*  ---------------------------- des_decipher ----------------------------  *
*                                                                       *
************************************************************************/

void des_decipher(const unsigned char *ciphertext, unsigned char *plaintext,
   const unsigned char *key) {

   des_main(ciphertext, plaintext, key, decipher);

   return;

}

static int des_linear_analysis_stage1(const unsigned int *rounds) {

   unsigned char input_bit_a = 0;
   unsigned char input_bit_b = 0;

   unsigned char output_bit_a = 0;
   unsigned char output_bit_b = 0;

   unsigned char      mask;

   int                i,j,k,l;

   for (i = 0; i < (input_bit_a % 8); i++)
   {
      input_bit_a = input_bit_a << 1;

      for (j = 0; j < (input_bit_b % 8); j++)
      {
```

```
          input_bit_b = input_bit_b << 1;

          for (k = 0; k < (output_bit_a % 8); k++)
          {
             output_bit_a = output_bit_a << 1;

             for (l = 0; l < (output_bit_b % 8); l++)
             {
                output_bit_b = output_bit_b << 1;

             }
          }
       }
    }

}

static unsigned char *keyrecovery des_linear_analysis_stage2(const unsigned int *rounds, co
nst unsigned char *ciphertexts[]) {
  return 0;
}
```

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "encrypt.h"

int main(int argc, char **argv) {

unsigned char      destmp[8],
                   desptx[8],
                   desctx[8],
                   deskey[8];

int                i;

/*****************************************************************************
*                                                                            *
*  DES-TOOLS
*
*  -r     Number of rounds
*  -z     Use zeroed key
*  -n     No permutations
*  -gfni  Use Galois Field New Instruction
*  -a     Analyze (determine linear approximation)
*  -v     Verbose
*                                                                            *
*****************************************************************************/

static struct CMD_OPTIONS {
        int f_ROUNDS            = FALSE;
        int f_ZEROED_KEY;       = FALSE;
        int f_NO_PERMUTATIONS   = FALSE;
        int f_USE_GFNI          = FALSE;
        int f_ANALYZE           = FALSE;
        int f_VERBOSE           = FALSE;
}

CMD_OPTIONS.f_ROUNDS = FASLE;
CMD_OPTIONS.f_ZEROED_KEY = FALSE;
CMD_OPTIONS.f_NO_PERMUTATIONS  = FALSE;
CMD_OPTIONS.f_USE_GFNI         = FALSE;
CMD_OPTIONS.f_ANALYZE          = FALSE;
CMD_OPTIONS.f_VERBOSE          = FALSE;

destmp[0] = 0xa9;
destmp[1] = 0x10;
destmp[2] = 0x11;
destmp[3] = 0x38;
destmp[4] = 0x93;
destmp[5] = 0xca;
destmp[6] = 0xb4;
destmp[7] = 0xa1;

deskey[0] = 0x01;
deskey[1] = 0x1f;
deskey[2] = 0x01;
deskey[3] = 0x1f;
deskey[4] = 0x01;
deskey[5] = 0x0e;
deskey[6] = 0x01;
deskey[7] = 0x0e;

fprintf(stdout, "DES-TOOLS\n");

   int opt;

   // put ':' in the starting of the
   // string so that program can
   //distinguish between '?' and ':'
```

```
   while((opt = getopt(argc, argv, â\200\234:if:lrxâ\200\235)) != -1)
   {
      switch(opt)
      {
         case 'r':
               CMD_OPTIONS.f_ROUNDS = TRUE;
               printf(â\200\234rounds: %c\nâ\200\235, opt);
               break;
         case 'z':
               CMD_OPTIONS.f_ZEROED_KEY = TRUE;
               printf(â\200\234zeroed key: %c\nâ\200\235, opt);
               break;
         case 'n':
               CMD_OPTIONS.f_NO_PERMUTATIONS = TRUE;
               printf(â\200\234no permutations: %c\nâ\200\235, opt);
               break;
         case 'gfni':
               printf(â\200\234use GFNI: %s\nâ\200\235, optarg);
               break;
         case 'a':
               printf(â\200\234Apply Linear Analysis : %s\nâ\200\235, optarg);
               break;
         case 'v':
               printf("Verbose": %s\nâ\200\235, optarg);
               break;
         case ':':
               printf(â\200\234option needs a value\nâ\200\235);
               break;
         case '?':
               printf(â\200\234unknown option: %c\nâ\200\235, optopt);
               break;
      }
   }

   // optind is for the extra arguments
   // which are not parsed
   for(; optind < argc; optind++){
      printf(â\200\234extra arguments: %s\nâ\200\235, argv[optind]);
   }



fprintf(stdout, "Using key: %02x %02x %02x %02x %02x %02x %02x %02x\n",
   deskey[0], deskey[1], deskey[2], deskey[3], deskey[4], deskey[5],
   deskey[6], deskey[7]);

des_encipher(destmp, desctx, deskey);

fprintf(stdout, "plaintext: %02x %02x %02x %02x %02x %02x %02x %02x\n",
   destmp[0], destmp[1], destmp[2], destmp[3], destmp[4], destmp[5],
   destmp[6], destmp[7]);

fprintf(stdout, "ciphertext: %02x %02x %02x %02x %02x %02x %02x %02x\n",
   desctx[0], desctx[1], desctx[2], desctx[3], desctx[4], desctx[5],
   desctx[6], desctx[7]);

return 0;

}
```

```
/*****************************************************************************
*                                                                            *
*  ex-1.c                                                                    *
*  ======                                                                    *
*                                                                            *
*  Description: Illustrates data encryption (see Chapter 15).               *
*                                                                            *
*****************************************************************************/

#include <stdio.h>
#include <string.h>

#include "encrypt.h"

/*****************************************************************************
*                                                                            *
*  ------------------------------ main ------------------------------------  *
*                                                                            *
*****************************************************************************/

int main(int argc, char **argv) {

unsigned char       destmp[8],
                    desptx[8],
                    desctx[8],
                    deskey[8];

Huge                rsatmp,
                    rsaptx,
                    rsactx;

RsaPubKey           rsapubkey;
RsaPriKey           rsaprikey;

int                 i;

/*****************************************************************************
*                                                                            *
*  Encipher some data using DES.                                             *
*                                                                            *
*****************************************************************************/

fprintf(stdout, "Enciphering with DES\n");

destmp[0] = 0xa9;
destmp[1] = 0x10;
destmp[2] = 0x11;
destmp[3] = 0x38;
destmp[4] = 0x93;
destmp[5] = 0xca;
destmp[6] = 0xb4;
destmp[7] = 0xa1;

deskey[0] = 0x01;
deskey[1] = 0x1f;
deskey[2] = 0x01;
deskey[3] = 0x1f;
deskey[4] = 0x01;
deskey[5] = 0x0e;
deskey[6] = 0x01;
deskey[7] = 0x0e;

fprintf(stdout, "Before enciphering\n");

fprintf(stdout, "destmp: %02x %02x %02x %02x %02x %02x %02x %02x\n",
   destmp[0], destmp[1], destmp[2], destmp[3], destmp[4], destmp[5],
   destmp[6], destmp[7]);
```

```
fprintf(stdout, "deskey: %02x %02x %02x %02x %02x %02x %02x %02x\n",
   deskey[0], deskey[1], deskey[2], deskey[3], deskey[4], deskey[5],
   deskey[6], deskey[7]);

des_encipher(destmp, desctx, deskey);

fprintf(stdout, "After enciphering\n");

fprintf(stdout, "destmp: %02x %02x %02x %02x %02x %02x %02x %02x\n",
   destmp[0], destmp[1], destmp[2], destmp[3], destmp[4], destmp[5],
   destmp[6], destmp[7]);

fprintf(stdout, "desctx: %02x %02x %02x %02x %02x %02x %02x %02x\n",
   desctx[0], desctx[1], desctx[2], desctx[3], desctx[4], desctx[5],
   desctx[6], desctx[7]);

fprintf(stdout, "Before deciphering\n");

fprintf(stdout, "desctx: %02x %02x %02x %02x %02x %02x %02x %02x\n",
   desctx[0], desctx[1], desctx[2], desctx[3], desctx[4], desctx[5],
   desctx[6], desctx[7]);

fprintf(stdout, "deskey: %02x %02x %02x %02x %02x %02x %02x %02x\n",
   deskey[0], deskey[1], deskey[2], deskey[3], deskey[4], deskey[5],
   deskey[6], deskey[7]);

des_decipher(desctx, desptx, deskey);

fprintf(stdout, "After deciphering\n");

fprintf(stdout, "desctx: %02x %02x %02x %02x %02x %02x %02x %02x\n",
   desctx[0], desctx[1], desctx[2], desctx[3], desctx[4], desctx[5],
   desctx[6], desctx[7]);

fprintf(stdout, "desptx: %02x %02x %02x %02x %02x %02x %02x %02x\n",
   desptx[0], desptx[1], desptx[2], desptx[3], desptx[4], desptx[5],
   desptx[6], desptx[7]);

/*************************************************************************
*                                                                        *
*  Encipher some data using RSA.
*  (code rem0ved)                                                *
*                                                                        *
*************************************************************************/

return 0;

}
```

```
/* RC5REF.C -- Reference implementation of RC5-32/12/16 in C.        */
/* Copyright (C) 1995 RSA Data Security, Inc.                        */
typedef unsigned long WORD;       /* Should be 32-bit = 4 bytes
*/
#define w        32              /* word size in bits                 */
#define r        12              /* number of rounds                  */
#define b        16              /* number of bytes in key            */
#define c         4              /* number  words in key = ceil(8*b/w)*/
#define t        26              /* size of table S = 2*(r+1) words   */
WORD S[t];                       /* expanded key table                */
WORD P = 0xb7e15163, Q = 0x9e3779b9;  /* magic constants              */
/* Rotation operators. x must be unsigned, to get logical right shift*/
#define ROTL(x,y) (((x)<<(y&(w-1))) | ((x)>>(w-(y&(w-1)))))
#define ROTR(x,y) (((x)>>(y&(w-1))) | ((x)<<(w-(y&(w-1)))))

void RC5_ENCRYPT(WORD *pt, WORD *ct) /* 2 WORD input pt/output ct    */
{ WORD i, A=pt[0]+S[0], B=pt[1]+S[1];
  for (i=1; i<=r; i++)
    { A = ROTL(A^B,B)+S[2*i];
      B = ROTL(B^A,A)+S[2*i+1];
    }
  ct[0] = A; ct[1] = B;
}

void RC5_DECRYPT(WORD *ct, WORD *pt) /* 2 WORD input ct/output pt    */
{ WORD i, B=ct[1], A=ct[0];
  for (i=r; i>0; i--)
    { B = ROTR(B-S[2*i+1],A)^A;
      A = ROTR(A-S[2*i],B)^B;
    }
  pt[1] = B-S[1]; pt[0] = A-S[0];
}

void RC5_SETUP(unsigned char *K) /* secret input key K[0...b-1]      */
{  WORD i, j, k, u=w/8, A, B, L[c];
   /* Initialize L, then S, then mix key into S */
   for (i=b-1,L[c-1]=0; i!=-1; i--) L[i/u] = (L[i/u]<<8)+K[i];
   for (S[0]=P,i=1; i<t; i++) S[i] = S[i-1]+Q;
   for (A=B=i=j=k=0; k<3*t; k++,i=(i+1)%t,j=(j+1)%c)   /* 3*t > 3*c */
     { A = S[i] = ROTL(S[i]+(A+B),3);
       B = L[j] = ROTL(L[j]+(A+B),(A+B));
     }
}
```