```
/*
 * The MIT License (MIT)
 *
 * Copyright (c) 2018 Zhou Zhi Gang
 * Email: keorapetse.finger@yahoo.com
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
#include <stdio.h>
#include <malloc.h>
#include "twofish.h"
#include "tables.h"

#define xor(g,r)    (g^r)                   /* Xor operation */
#define ror(g,n)    ((g>>n)|(g<<(32-n)))    /* Rotate right  */
#define rol(g,n)    ((g<<n)|(g>>(32-n)))    /* Rotate left   */
#define nxt(g,r)    (*(g+r))                /* Get next byte */
#define LITTILE_ENDIAN
#ifdef  LITTILE_ENDIAN
#define unpack(g,r) ((g>>(r*8))&0xff)                               /* Extracts a byte from a word.  */
#define pack(g)     ((*(g))|(*(g+1)<<8)|(*(g+2)<<16)|(*(g+3)<<24))  /* Converts four byte to a word. */
#endif
#define pad_factor(g) \
        if (g<16)       n = 16; \
        else if (g<24)  n = 24; \
        else if (g<32)  n = 32;
#define rsm(i,a,b,c,d,e,f,g,h)  \
        gf(nxt(tf_key->k,r*8),a,0x14d)^gf(nxt(tf_key->k,r*8+1),b,0x14d)^\
        gf(nxt(tf_key->k,r*8+2),c,0x14d)^gf(nxt(tf_key->k,r*8+3),d,0x14d)^\
        gf(nxt(tf_key->k,r*8+4),e,0x14d)^gf(nxt(tf_key->k,r*8+5),f,0x14d)^\
        gf(nxt(tf_key->k,r*8+6),g,0x14d)^gf(nxt(tf_key->k,r*8+7),h,0x14d)
#define u(x,a)\
        x[0] = unpack(a,0); \
        x[1] = unpack(a,1); \
        x[2] = unpack(a,2); \
        x[3] = unpack(a,3);
#define release(a,b,c)  { free(a); free(b);free(c); }
#ifdef  TWOFISH
typedef struct key_t
{
    uint8_t len;
    uint8_t *k;
}key_t;
typedef struct subkey_t
{
    uint8_t len;
    uint8_t s[4][4];
    uint8_t me[4][4];
    uint8_t mo[4][4];
}subkey_t;
```

```c
#endif
/*
 * Twofish Expand Key Function
 *
 * Description:
 *
 * @param   s
 * @param   len
 * @usage
 * {@code}
 */
key_t* Twofish_expand_key(uint8_t *s, uint32_t len);
/*
 * Twofish Galois Field Multiplication Function
 *
 * Description:
 *
 * @param   x
 * @param   y
 * @param   m
 * @usage
 * {@code}
 */
uint8_t gf(uint8_t x, uint8_t y, uint16_t m);
/*
 * Twofish Generate Subkeys Function
 *
 * Description:
 *
 * @param   tf_key
 * @usage
 * {@code}
 */
subkey_t* Twofish_generate_subkey(key_t* tf_key);
/*
 * Twofish Generate Subkeys Function
 *
 * Description:
 *
 * @param   x[]
 * @param   y[]
 * @param   s
 * @param   stage
 * @usage
 * {@code}
 */
void Twofish_h(uint8_t x[],  uint8_t y[], uint8_t s[][4], int stage);
/*
 * Twofish MDS Multiply Function
 *
 * Description:
 *
 * @param   y[]
 * @param   out[]
 * @usage
 * {@code}
 */
void Twofish_mds_mul(uint8_t y[],  uint8_t out[]);
/*
 * Twofish MDS Multiply Function
 *
 * Description:
 *
 * @param   tf_twofish
 * @param   tf_subkey
 * @param   p
 * @param   k
 * @usage
 * {@code}
 */
```

```c
twofish_t* Twofish_generate_ext_k_keys(twofish_t* tf_twofish, subkey_t *tf_subkey,uint32_t p, uint8_t k);
/*
 * Twofish MDS Multiply Function
 *
 * Description:
 *
 * @param   tf_twofish
 * @param   tf_subkey
 * @param   k
 * @usage
 * {@code}
 */
twofish_t* Twofish_generate_ext_s_keys(twofish_t* tf_twofish, subkey_t *tf_subkey, uint8_t k);
/*
 * Twofish f Function
 *
 * Description:
 *
 * @param   tf_twofish
 * @param   r
 * @param   r0, r1
 * @param   f0, f1
 * @usage
 * {@code}
 */
void Twofish_f(twofish_t* tf_twofish, uint8_t r,uint32_t r0, uint32_t r1, uint32_t* f0, uint32_t* f1);
/*
 * Twofish g Function
 *
 * Description:
 *
 * @param   tf_twofish
 * @param   x
 * @usage
 * {@code}
 */
uint32_t Twofish_g(twofish_t* tf_twofish, uint32_t x);

twofish_t* Twofish_setup(uint8_t *s, uint32_t len)
{
    /* Expand the key if necessary. */
    key_t* tf_key = Twofish_expand_key(s, len);

    /* Generate subkeys: s and k */
    subkey_t *tf_subkey = Twofish_generate_subkey(tf_key);

    /* Generate 40 K keys */
    twofish_t* tf_twofish = (twofish_t*)malloc(sizeof(twofish_t));
    tf_twofish = Twofish_generate_ext_k_keys(tf_twofish,tf_subkey,0x01010101,(tf_key->len/8));
     /* Generate 4x256 S keys */
    tf_twofish = Twofish_generate_ext_s_keys(tf_twofish,tf_subkey,(tf_key->len/8));

    /* Free memory */
    release(tf_key->k, tf_key, tf_subkey);

    return tf_twofish;
}

void Twofish_encryt(twofish_t* tf_twofish, uint8_t *data, uint8_t *cypher)
{
    uint32_t r0, r1, r2, r3, f0, f1, c2,c3;
    /* Input Whitenening */
    r0 = tf_twofish->k[0]^pack(data);
    r1 = tf_twofish->k[1]^pack(data+4);
    r2 = tf_twofish->k[2]^pack(data+8);
    r3 = tf_twofish->k[3]^pack(data+12);

    /* The black box */
    for (int i=0; i<16;++i)
    {
        Twofish_f(tf_twofish, i, r0, r1, &f0, &f1);
        c2 = ror((f0 ^r2), 1);
        c3 = (f1^rol(r3,1));
        /* swap */
        r2 = r0;
        r3 = r1;
        r0 = c2;
        r1 = c3;
    }

    /* Output Whitening */
    c2 = r0;
    c3 = r1;
    r0 = tf_twofish->k[4]^r2;
    r1 = tf_twofish->k[5]^r3;
    r2 = tf_twofish->k[6]^c2;
    r3 = tf_twofish->k[7]^c3;

    for (int i=0;i<4;++i)
    {
        cypher[i]   = unpack(r0,i);
        cypher[i+4] = unpack(r1,i);
        cypher[i+8] = unpack(r2,i);
        cypher[i+12]= unpack(r3,i);
    }
}

void Twofish_decryt(twofish_t* tf_twofish, uint8_t *cypher, uint8_t *data)
{
    uint32_t r0, r1, r2, r3, f0, f1, c2,c3;
    /* Input Whitenening */
    r0 = tf_twofish->k[4]^pack(cypher);
    r1 = tf_twofish->k[5]^pack(cypher+4);
    r2 = tf_twofish->k[6]^pack(cypher+8);
    r3 = tf_twofish->k[7]^pack(cypher+12);

    /* The black box */
    for (int i=15; i >= 0;--i)
    {
        Twofish_f(tf_twofish, i, r0, r1, &f0, &f1);
        c2 = (rol(r2,1)^f0);
        c3 = ror((f1^r3),1);
        /* swap */
        r2 = r0;
        r3 = r1;
        r0 = c2;
        r1 = c3;
    }

    /* Output Whitening */
    c2 = r0;
    c3 = r1;
    r0 = tf_twofish->k[0]^r2;
    r1 = tf_twofish->k[1]^r3;
    r2 = tf_twofish->k[2]^c2;
    r3 = tf_twofish->k[3]^c3;

    for (int i=0;i<4;++i)
    {
        data[i]   = unpack(r0,i);
        data[i+4] = unpack(r1,i);
        data[i+8] = unpack(r2,i);
        data[i+12]= unpack(r3,i);
    }
}
```

```

void Twofish_f(twofish_t* tf_twofish, uint8_t r,uint32_t r0, uint32_t r1, uint32_t* f0, uint32_t* f1)
{
    uint32_t t0, t1, o;
    t0 = Twofish_g(tf_twofish, r0);
    t1 = rol(r1, 8);
    t1 = Twofish_g(tf_twofish, t1);
    o = 2*r;
    *f0= (t0 + t1 + tf_twofish->k[o+8]);
    *f1= (t0 + (2*t1) + tf_twofish->k[o+9]);
}

twofish_t* Twofish_generate_ext_k_keys(twofish_t* tf_twofish, subkey_t * tf_subkey,uint32_t p, uint8_t k)
{
    uint32_t a, b, o;
    uint8_t x[4], y[4], z[4];
    for(int i=0;i<20;++i)
    {
        a = (2*i*p);
        b = (a+p);
        u(x,a);
        Twofish_h(x, y, tf_subkey->me, k);
        Twofish_mds_mul(y,z);
        a = pack(z);                        /* Convert four bytes z[4] to a word (a). */
        u(x,b);                             /* Convert a word (b) to four bytes x[4]. */
        Twofish_h(x, y, tf_subkey->mo, k);
        Twofish_mds_mul(y,z);
        b = pack(z);
        b = rol(b, 8);
        o = 2*i;
        tf_twofish->k[o] = ((a + b));
        tf_twofish->k[o+1] = rol(((a + (2*b))),9);
    }
    return tf_twofish;
}

twofish_t* Twofish_generate_ext_s_keys(twofish_t* tf_twofish, subkey_t * tf_subkey, uint8_t k)
{
    uint8_t x[4], y[4];
    for(int i=0;i<256;++i)
    {
        x[0] = x[1] = x[2] = x[3] = i;
        Twofish_h(x, y, tf_subkey->s, k);
        /* Special MDS multiplication */
        tf_twofish->s[0][i] = (gf(y[0], mds[0][0],0x169) |(gf(y[0],mds[1][0], 0x169)<< 8)|(gf(y[0], mds[2][0],0x169)<<16) |(gf(y[0], mds[3][0], 0x169) <<24));
        tf_twofish->s[1][i] = (gf(y[1], mds[0][1],0x169) |(gf(y[1],mds[1][1], 0x169)<< 8)|(gf(y[1], mds[2][1],0x169)<<16) |(gf(y[1], mds[3][1], 0x169) <<24));
        tf_twofish->s[2][i] = (gf(y[2], mds[0][2],0x169) |(gf(y[2],mds[1][2], 0x169)<< 8)|(gf(y[2], mds[2][2],0x169)<<16) |(gf(y[2], mds[3][2], 0x169) <<24));
        tf_twofish->s[3][i] = (gf(y[3], mds[0][3],0x169) |(gf(y[3],mds[1][3], 0x169)<< 8)|(gf(y[3], mds[2][3],0x169)<<16) |(gf(y[3], mds[3][3], 0x169) <<24));
    }
    return tf_twofish;
}

void Twofish_mds_mul(uint8_t y[],  uint8_t out[])
{
    out[0] = (gf(y[0], mds[0][0], 0x169)^gf(y[1], mds[0][1], 0x169)^gf(y[2], mds[0][2], 0x169)^gf(y[3], mds[0][3], 0x169) );
    out[1] = (gf(y[0], mds[1][0], 0x169)^gf(y[1], mds[1][1], 0x169)^gf(y[2],
mds[1][2], 0x169)^gf(y[3], mds[1][3], 0x169) );
    out[2] = (gf(y[0], mds[2][0], 0x169)^gf(y[1], mds[2][1], 0x169)^gf(y[2], mds[2][2], 0x169)^gf(y[3], mds[2][3], 0x169) );
    out[3] = (gf(y[0], mds[3][0], 0x169)^gf(y[1], mds[3][1], 0x169)^gf(y[2], mds[3][2], 0x169)^gf(y[3], mds[3][3], 0x169) );
}

uint32_t Twofish_g(twofish_t* tf_twofish, uint32_t x)
{
    return (tf_twofish->s[0][unpack(x, 0)]^tf_twofish->s[1][unpack(x, 1)]^tf_twofish->s[2][unpack(x, 2)]^tf_twofish->s[3][unpack(x, 3)]);
}

void Twofish_h(uint8_t x[],  uint8_t out[], uint8_t s[][4], int stage)
{
    uint8_t y[4];
    for (int j=0; j<4;++j)
    {
        y[j] = x[j];
    }

    if (stage == 4)
    {
        y[0] = q[1][y[0]] ^ (s[3][0]);
        y[1] = q[0][y[1]] ^ (s[3][1]);
        y[2] = q[0][y[2]] ^ (s[3][2]);
        y[3] = q[1][y[3]] ^ (s[3][3]);
    }
    if (stage > 2)
    {
        y[0] = q[1][y[0]] ^ (s[2][0]);
        y[1] = q[1][y[1]] ^ (s[2][1]);
        y[2] = q[0][y[2]] ^ (s[2][2]);
        y[3] = q[0][y[3]] ^ (s[2][3]);
    }

    out[0] = q[1][q[0][ q[0][y[0]] ^ (s[1][0])] ^ (s[0][0])];
    out[1] = q[0][q[0][ q[1][y[1]] ^ (s[1][1])] ^ (s[0][1])];
    out[2] = q[1][q[1][ q[0][y[2]] ^ (s[1][2])] ^ (s[0][2])];
    out[3] = q[0][q[1][ q[1][y[3]] ^ (s[1][3])] ^ (s[0][3])];
}

subkey_t* Twofish_generate_subkey(key_t* tf_key)
{
    int k, r, g;
    subkey_t *tf_subkey = (subkey_t*)malloc(sizeof(subkey_t));
    k = tf_key->len/8;                              /* k=N/64 */
    for(r=0; r<k;++r)
    {
        /* Generate subkeys Me and Mo */
        tf_subkey->me[r][0] = nxt(tf_key->k, r*8 + 0);
        tf_subkey->me[r][1] = nxt(tf_key->k, r*8 + 1);
        tf_subkey->me[r][2] = nxt(tf_key->k, r*8 + 2);
        tf_subkey->me[r][3] = nxt(tf_key->k, r*8 + 3);
        tf_subkey->mo[r][0] = nxt(tf_key->k, r*8 + 4);
        tf_subkey->mo[r][1] = nxt(tf_key->k, r*8 + 5);
        tf_subkey->mo[r][2] = nxt(tf_key->k, r*8 + 6);
        tf_subkey->mo[r][3] = nxt(tf_key->k, r*8 + 7);

        g=k-r-1;                                    /* Reverse order */
        /* Generate subkeys S using RS matrix */
        tf_subkey->s[g][0] = rsm(r, 0x01, 0xa4, 0x55, 0x87, 0x5a, 0x58, 0xdb, 0x9e);
        tf_subkey->s[g][1] = rsm(r, 0xa4, 0x56, 0x82, 0xf3, 0x1e, 0xc6, 0x68, 0xe5);
        tf_subkey->s[g][2] = rsm(r, 0x02, 0xa1, 0xfc, 0xc1, 0x47, 0xae, 0x3d, 0x19);
        tf_subkey->s[g][3] = rsm(r, 0xa4, 0x55, 0x87, 0x5a, 0x58, 0xdb, 0x9e, 0x03);
    }
    return tf_subkey;
}

key_t* Twofish_expand_key(uint8_t *s, uint32_t len)
{
    int n;
    pad_factor(len);
    key_t* tf_key = (key_t*)malloc(sizeof(key_t));
    uint8_t* ss = (uint8_t*)malloc(n);
    for (int g=0; g<n; ++g)
    {
        *(ss+g) = 0x00;
        if (g < len)
            *(ss+g) = *(s+g);
    }
    tf_key->k = ss;
    tf_key->len=n;
    return tf_key;
}

uint8_t gf(uint8_t x, uint8_t y, uint16_t m)
{
    uint8_t c, p = 0;
    for (int i=0; i<8; ++i)
    {
        if (y & 0x1)
            p ^= x;
        c = x & 0x80;
        x <<= 1;
        if (c)
            x ^= m;
        y >>= 1;
    }
    return p;
}

```